Interactive Visual Analysis of Software Structures

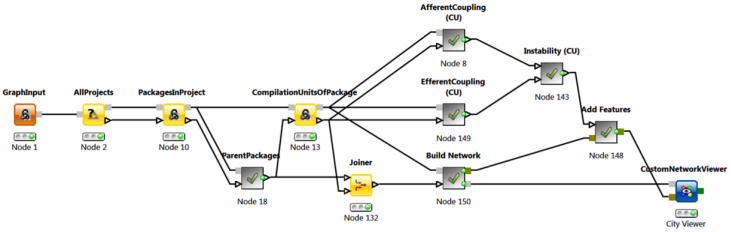
Vom Fachbereich Informatik
der Technischen Universität Kaiserslautern
zur Erlangung des akademischen Grades
Doktor der Ingenieurswissenschaften (Dr.-Ing.)
genehmigte Dissertation
von

Taimur Kausar Khan

Datum der wissenschaftlichen Aussprache: 17. Juli 2015

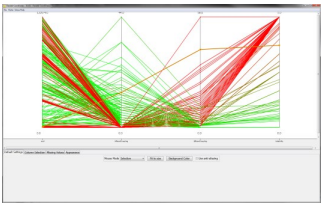
Dekan:	Prof. Dr. rer. nat. Klaus Schneider
Promotionsausschuss-Vorsitzender:	Prof. Dr.-Ing. Jens B. Schmitt
Berischterstatter:	Prof. Dr. Hans Hagen
	Prof. Dr. Achim Ebert
	Prof. Dr.-Ing. habil. Peter Liggesmeyer

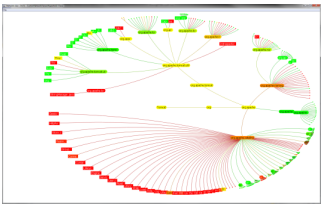
D 386

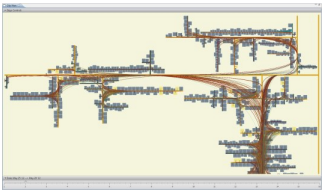


TAIMUR KHAN

INTERACTIVE VISUAL ANALYSIS OF SOFTWARE STRUCTURES







𝔗o my parents, my wife Heike, and my son Noah

Preface

I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family and wife. I would like to express my gratitude to all of them, realizing that those whom I owe the most I cannot thank enough, and that the things for which I am most grateful, I cannot put into words.

I would like to express my deepest gratitude to my advisors, Prof. Dr.-Ing. Peter Liggesmeyer and Prof. Dr. Achim Ebert, for their excellent guidance, care, patience, and providing me with an excellent atmosphere for doing research. It would be hard to overstate how much I benefited from their deep insight, their invariably correct intuitions, and their unwavering support. I cannot thank them adequately for all the effort they have made to understand me and to bring out the best in me. I thank them for their valuable suggestions and comments on this manuscript.

I would like to give a heartfelt thanks to Prof. Dr. Hans Hagen for being a mentor to me for the past six years. I will always be indebted to him for the chance to be a part of the International Research Training Group 1131 (IRTG) and for inviting me to write a Doctoral thesis. He has given me guidance and advice on a personal and professional level that has been instrumental in my growth as a research student preparing to be a professional. He consistently makes himself available to his students, to listen and encourage them to pursue their desired interests. While I can never thank him enough, I want him to know that I am truly grateful for his leadership by

example as he continuously demonstrates excellent qualities as a researcher, teacher, friend, and above all, his role as a mentor. I hope I can one day make an impact to someone else the way he has done to me.

I would like to thank Prof. Dr. Frederic McKenzie for being the one to put me on the path towards academic enlightenment. I still remember the excitement of working on my Masters and turning in my first paper, this was my first introduction to research. I am also grateful to Dr. Henning Barthel who let me experience the research of software visualization in the field and face practical issues beyond textbooks. The four years of collaboration, guidance, and support helped me understand what my ideas really were.

I would like to thank all my colleagues at the *Computer Graphics and HCI* and *Software Engineering: Dependability* groups for the rewarding discussions and intellectual exchanges, especially Daniel Engel, Mathias Hummel, and Sebastian Petsch. Furthermore, I would also like to thank Mady Gruys, Inga Scheler, and Roger Daneker for always being there and assisting me in the day-to-day mundane duties.

Finally, I would like to thank my wife Heike and my son Noah. Heike was always there cheering me up and stood by me through the good times and bad. Her love, support, and constant patience have taught me so much about sacrifice, discipline and compromise. Noah was born before this dissertation was completed and spent many days without my presence. I am deeply sorry for the time we spent apart.

Kaiserslautern, 20-Jul-2015 *Taimur Khan*

Abstract

Maintaining complex software systems tends to be a costly activity where software engineers spend a significant amount of time trying to understand the system's structure and behavior. As early as the 1980s, operation and maintenance costs were already twice as expensive as the initial development costs incurred. Since then these costs have steadily increased. The focus of this thesis is to reduce these costs through novel interactive exploratory visualization concepts and to apply these modern techniques in the context of services offered by software quality analysis.

Costs associated with the understanding of software are governed by specific features of the system in terms of different domains, including re-engineering, maintenance, and evolution. These features are reflected in software measurements or inner qualities such as *extensibility, reusability, modifiability, testability, compatability, or adatability.* The presence or absence of these qualities determines how easily a software system can conform or be customized to meet new requirements. Consequently, the need arises to monitor and evaluate the qualitative state of a software system in terms of these qualities. Using metrics-based analysis, production costs and quality defects of the software can be recorded objectively and analyzed.

In practice, there exist a number of free and commercial tools that analyze the inner quality of a software system through the use of software metrics. However, most of these tools focus on software data mining and metrics (computational analysis) and only a few support visual analytical reasoning. Typically, computational analysis tools generate data and software visualization tools facilitate the

exploration and explanation of this data through static or interactive visual representations. Tools that combine these two approaches focus only on well-known metrics and lack the ability to examine user defined metrics. Further, they are often confined to simple visualization methods and metaphors, including charts, histograms, scatter plots, and node-link diagrams.

The goal of this thesis is to develop methodologies that combine computational analysis methods together with sophisticated visualization methods and metaphors through an interactive visual analysis approach. This approach promotes an iterative knowledge discovery process through multiple views of the data where analysts select features of interest in one of the views and inspect data items of the select subset in all of the views. On the one hand, we introduce a novel approach for the visual analysis of software measurement data that captures complete facts of the system, employs a flow-based visual paradigm for the specification of software measurement queries, and presents measurement results through integrated software visualizations. This approach facilitates the on-demand computation of desired features and supports interactive knowledge discovery – the analyst can gain more insight into the data through activities that involve: building a mental model of the system; exploring expected and unexpected features and relations; and generating, verifying, or rejecting hypothesis with visual tools. On the other hand, we have also extended existing tools with additional views of the data for the presentation and interactive exploration of system artifacts and their inter-relations.

Contributions of this thesis have been integrated into two different prototype tools. First evaluations of these tools show that they can indeed improve the understanding of large and complex software systems.

Kurzfassung

Die Pflege komplexer Software-Systeme ist eine zunehmend kostspielige Tätigkeit, bei der Software-Ingenieure einen Großteil ihrer Zeit damit verbringen, Struktur und Verhalten der Systeme zu verstehen. Bereits in den 80er Jahren waren Betriebs- und Wartungskosten doppelt so teuer wie die anfänglichen Entwicklungskosten größerer Softwaresysteme. Seither ist der Anteil dieser Kosten an den Gesamtkosten stetig gestiegen. Der Schwerpunkt dieser Arbeit ist es, diese Kosten durch neuartige interaktive explorative Visualisierungskonzepte zu reduzieren und diese modernen Techniken im Kontext der von Software-Qualitätsanalyse angebotenen Dienste zu integrieren.

Kosten, die mit dem Verständnis der Software verbunden sind, unterliegen bestimmten qualitativen Eigenschaften des Systems in Bezug auf verschiedene Aspekte des Gesamtlebenszyklusses, einschließlich des Re-Engineering, der Wartung, und Entwicklung. Diese Eigenschaften werden bewertet durch Software-Metriken, die sogenannte innere Qualitäten (wie Erweiterbarkeit, Wiederverwendbarkeit, Änderbarkeit, Testbarkeit, Kompatibilität oder Anpassbarkeit an unterschiedliche Umgebungen) messen. Das Vorhandensein oder Fehlen dieser inneren Qualitäten bestimmt, wie leicht ein Softwaresystem neuen Anforderungen gerecht wird oder gemacht werden kann. Um Wartungs- und Betriebskosten zu reduzieren, ist es notwendig, die inneren Qualitäten eines Software-Systems zu gewährleisten. Folglich ergibt sich die Notwendigkeit zur Überwachung und Bewertung des qualitativen Zustand eines Softwaresystems. Mittels metriken-basierter Analyse können Produktionskosten und Qualitätsmängel der Software objektiv erfasst und analysiert werden.

Für der praktischen Einsatz existieren eine Reihe von kostenlosen und kommerziellen Tools, die die innere Qualität eines Softwaresystems durch den Einsatz von Software-Metriken analysieren. Diese konzentrieren sich jedoch meist auf Software-Data-Mining und Metriken (Automatische Datenanalyse) und nur wenige bieten Unterstützung bei der visuellen Analyse. Typischerweise generieren automatische Datenanalysetools Daten und Software-Visualisierungstools ermöglichen die Untersuchung und Erklärung dieser Daten durch statische oder interaktive visuelle Darstellungen. Werkzeuge, die diese beiden Ansätze kombinieren, konzentrieren sich nur auf bekannte Metriken und unterstützen nicht die Untersuchung von benutzerdefinierten Metriken. Darüber hinaus sind diese oft auf da Einsatz einfach zu integrierender Visualisierungsmethoden und Metaphern wie beispielsweise Charts, Histogramme, Streudiagramme und Node-Link-Diagramme beschränkt.

Das Ziel dieser Arbeit ist es, Data Mining Methoden und Softwaremetriken zusammen mit anspruchsvollen Visualisierungsmethoden und -metaphern zu einem interaktiven visuellen Analyseansatz zu kombinieren. Dieser Ansatz fördert eine iterative Schaffung von Wissen durch mehrere Ansichten der Daten, in denen Analysten beispielsweise interessante Merkmale in einer der Ansichten auswählen und die Datenelemente der ausgewählten Untergruppe in anderen Ansichten weiter inspizieren können. Zunächst stellen wir einen neuen Ansatz für die visuelle Analyse von Software-Messdaten vor. Dieser umfasst eine integrierte Software-Visualisierung, die die vollständigen Fakten des Systems aufzeigt, sowie anhand eines Flussdiagrammes visuelle Spezifikationen von Datenbankabfragen der Software-Messdaten ermöglicht und die zugehörigen Messdaten präsentiert. Dieser Ansatz ermöglicht die Berechnung und Bewertung von Softwareeigenschaften somit gewinnt das Softwaresystem, unterstützt durch den Aufbau eines mentalen Modells, der Erkundung erwarteter und unerwarteter Eigenschaften und Beziehungen, sowie der Überprüfung von Hypothesen. Ein weiteres Ziel dieser Arbeit ist es, vorhandene Werkzeuge mit zusätzlichen Ansichten der Daten bezüglich der Darstellung und interaktiven Exploration von Systemartefakten zu ergänzen, sowie deren Wechselbeziehungen zu erweitern.

Beiträge dieser Arbeit wurden in zwei verschiedene prototypische Werkzeuge integriert. Erste Evaluierungen dieser Werkzeuge zeigen, dass deren Auswirkungen tatsächlich das Verständnis großer und komplexer Softwaresysteme verbessern können.

Contents

Appendix

List of Figures

List of Tables

1

Introduction and Overview

" *You need statistics to describe data, but then visualization to see it in context* "
– Andy Kirk [1]

This thesis presents novel methods for the interactive visual analysis of software systems. In particular, we present various approaches suitable for the analysis of data from a wide range of problem domains, including re-engineering, software maintenance, and software evolution. Visual analysis can enhance the software analysts' cognitive ability to comprehend analysis results and can enable the formation of valuable knowledge from raw data artifacts. It facilitates the *sense-making* process through the processing of visual information to detect interesting structures and relationships in the data such as patterns, trends, and anomalies. In this chapter, we first provide a brief description of the problem domain: *the analysis of software systems*. Next, we present a short introduction of the proposed methodologies: *Visualization and Interactive Visual Analysis (IVA)*. Finally, we outline the main contributions of this work and provide an overview of the structure of this thesis.

[1] Founder of Visualizing Data Ltd and author of Visualisingdata.com

1.1 Analysis of Software Systems

Software systems nowadays are a integral part of our daily lives. They can be found all around us in conventional desktop and internet applications or in almost all technical devices, from mobile phones to traffic lights. In addition to becoming more pervasive, they have also become more complex, large, and sophisticated. Consequently, more resources are allocated to make sure that the software is correct, robust, reliable, and of high quality.

The field of software engineering aims to address these concerns through systematic and disciplined activities. The IEEE standard 601.12 defines software engineering as "the application of systematic, disciplined, quantifiable approaches to the development, operation, and maintenance of software". Typically these approaches can be categorized into a hierarchy of activities, "ranging from requirements gathering, specification, and design to implementation, debugging, testing, and maintenance" [14]. Although these activities or more specifically software engineering sub-disciplines are quite varied, they all have one common goal; to better understand the macroscopic properties of software systems to make well-informed decisions about re-engineering, maintenance, software evolution, etc. [15].

This need to understand software systems from different perspectives coupled with the size and complexity of modern software systems makes it a towering task, so much so that professional software developers spend more time analyzing code than developing it [16]. Further, this understanding is convoluted due to various ways of structuring source code [17, 18] (i.e. as a file hierarchy, a network of components, a set of design patterns or aspects, etc.), the complexity of inter-hierarchy relations [19], and the complexity of software evolution [20]. In general, understanding software is hard because it is large, complex, abstract, and changing [21].

In practice, there are three main categories of analysis tools and techniques: *static code analysis*, that reviews "static" (non-executing) source code to highlight possible vulnerabilities [22] such as syntactic pattern matching, type systems, data-flow analysis, and abstract implementation; *dynamic code analysis*, that detects run-time errors such as memory leaks and null pointer errors [23]; and *hybrid analysis*, that integrates static and dynamic analysis approaches [24, 25]. This

thesis mostly is concerned with static analysis, however, our approach can easily be extended to encompass the other two categories. The interested reader may refer to a report compiled by Larsen et al. [26] that lists the state of the art software analysis tools and techniques.

Most software analysis tools follow a database approach due to the sheer size of software facts and the inherent structure of software. Much like a database, software contains a set of entities that range from components, files, classes, functions to code lines and expressions; and relationships, such as call, containment, and build dependencies [14]. Further, entities and relationships have various numerical, ordinal, or textual attributes, such as types of data access, quality and complexity metrics, and source code. In practice, analysis tools extract syntax, dependency, and execution facts from software [27–29] and refine these facts into measurements, such as code readability, cohesion, and coupling [28], or higher-level artifacts like design patterns [27] or code smells [29]. Finally, the raw data generated is typically understood through computational analysis methods such as data mining, exploration, and presentation [30, 31].

While computational methods are essential, they are not always adequate. They typically require problems (queries) to be precisely defined from the beginning [32] and lack the means to exploratively analyze the underlying raw data. Further, the results of automated analysis of complex problems can be difficult to understand. It is not always intuitive to explore trends, patterns, relations, and dependencies in data through statistical aggregates. Usually, one requires knowledge of properties of interest and data patterns to compute useful statistical aggregates. Such knowledge is difficult to attain and using common statistical aggregates without it can lead to important data features remaining completely hidden [33]. Further, this can lead to situations where traditional data mining techniques may fail to find the most basic data features [34]. However, where applying computational methods alone fails, *Visualization and IVA* can support the analysis and knowledge generation from complex software analysis data [35].

1.2 Visualization and Interactive Visual Analysis

The field of Visualization or more specifically *Information Visualization* has emerged from research in human-computer interaction, computer science, graphics, visual design, psychology, and business methods. It has been defined by Card et al. [36] as "the use of computer-supported, interactive, visual representations of data to amplify cognition". In recent times, it has been increasingly applied as a "critical component in scientific research, digital libraries, data mining, financial data analysis, market studies, manufacturing production control, and drug discovery" [37].

There are three major application goals or stages for visualization [38]. In the first stage, *Visual Exploration*, new and unknown data characteristics are visually investigated [39]. The analyst tries to discover trends, patterns, clusters, outliers, and relationships in the data; and formulate hypotheses about the relationships and dependencies in the data [38]. In the second stage, *Visual Analysis*, a goal-oriented analysis process is used to confirm or reject a hypothesis [40]. Finally in the third stage, *Presentation*, knowledge gained in analysis is communicated and disseminated to domain experts, decision makers, or the general public [40, 41]. An important lesson that is learned from research but often forgotten is that visualization must be tailored with respect to different users and tasks [38, 42].

There is even a large subarea of Information Visualization, called *Software Visualization*, that is dedicated to support the understanding of software systems [43–45], the analysis and exploration of software systems and their anomalies [14, 46–48], and their development and evolution [49–51]. Similar to the lesson learned from Information Visualization, the work of Telea et al. [45] indicates that software visualization need to focus more on tailored solutions.

On the other hand, IVA has evolved out of the fields of information and scientific visualization [52]. It is a multi-disciplinary approach that combines computational and interactive visual data analysis methods. Whereas computational methods require analysts to explicitly formulate their questions [34], IVA promotes iterative knowledge discovery. It typically uses several distinct views to show different aspects of the data set [53, 54]. These views can be commonly used representations (i.e., histograms, scatter plots, parallel coordinates, etc.) or even custom-built software visualizations. Regardless of the number or types of views, IVA systems have one common aspect;

they allow analysts to select *features of interest* in one of the views and highlight data items of the selected subset in all the views [32]. This *brushing and linking* approach enables analysts to emphasize and correlate different perspectives on the features of interest.

In literature and practice, there are many data mining tools and techniques that are used in software engineering [27, 30, 31]. Additionally, the entire field of software visualization is dedicated to visual methods and tools for software understanding [43, 44]. In contrast, there is very limited work that combines computational analysis methods together with software visualization [14]. In this thesis, we look to bridge this gap through various IVA approaches.

1.3 Contribution

This thesis is concerned with the interactive visual analysis of the static aspects of software structures. The methodology presented in this thesis is suitable for the analysis of data from a wide range of domains, including software re-engineering, software maintenance, and software evolution. These principles have been successfully integrated into a framework that combines computational analysis and IVA approaches [55, 56], frameworks that extend existing software analysis tools with IVA techniques [57–60], and a framework that applies IVA on a large High-Resolution (Hi-Res) display for collaborative software analysis tasks [61].

The main contributions of this thesis advance certain aspects in the visual analysis of software structures. We introduce a novel approach for the *Visual Analysis of Software Measurement Data* that consists of a data model that captures complete facts of the source code, a flow-based paradigm for the visual specification of software metric queries, and integrated software visualizations. We introduce means of extending existing analysis tools for the *Visual Analysis of Software Architecture Evolution* and the *Visual Analysis of Software Architectural Relations*. A complementary contribution, *Visual Analysis in a Collaborative Environment*, extends our ideas for collaborative software analysis tasks. A summary of these contributions is provided below with more details following in subsequent chapters.

1.3.1 Visual Analysis of Software Measurement Data

There is currently a large gap between software data mining and metrics tools (computational analysis) and software visualization. Typically the computational analysis part, consisting of software fact mining and metrics computation, and the visualization part, consisting of interactive software visualization, are separated in practice through different tools. Interesting data is extracted through computational methods using one tool and visualized through another. In contrast, we present a novel methodology of combining the two in a flow-based approach.

While most computational analysis tools employ relational databases, we advocate for a graph database approach. Due to the large amounts of analysis data generated by extracting source code facts, traditional approaches either support a limited number of well-known metrics or trade the amount of source code details stored for performance. Instead, graph databases provide a means to perform a complete analysis of large software system at interactive rates. In this regards, we present a data model that captures full details of the source code and is designed for optimized graph traversals for software analysis queries.

Further, current computational analysis tools are designed to either support only well-known metrics or are too complicated to use in generating custom software metrics. The analyst requires extensive knowledge of the underlying data schemata and the relevant querying language. In our work, we alleviate the analyst from this burden through an interactive visual workflow modeling approach where the focuses is on visual elements, their configurations, and interconnectivity rather than the data ontology and querying language.

In order to test and validate our ideas in the wild, we have developed a prototype tool called Visual Specification of Metrics (VIMETRIK). Our initial studies show that users with diverse backgrounds and no knowledge of the data ontology or the querying mechanisms could not only interactively generate software measurements. They could also visualize them in an integrated environment that combines computational analysis and software visualization.

1.3.2 Visual Analysis of Software Architecture Evolution

Analyzing the architecture of a software system plays an important role in the context of software maintenance and evolution. The structure of a software system is analyzed to explain how a system has evolved to its present state and to predict its future development. In this regards, current tools are confined with easy to integrate visualization techniques such as node-link diagrams that lack the sophistication to handle informative large-scale software architecture evolution visualization. However, there a number of solutions proposed in the research of software visualization that have not made it to the mainstream. In our work, we have looked to bridge this gap by extending an existing software architecture analysis tool through an IVA means that incorporates state-of-the-art software visualization.

Our methodology consists of augmenting traditional views of the data with additional views and apply "brushing and linking" to facilitate the analyst in making appropriate correlations and understanding the structural evolution of software structures. In our work, we developed the Evolving CITY (eCITY) tool that extends a "software city layout" proposed by Steinbrückner et al. [51] that explicitly takes development history of software systems into account and makes it directly available in the layouts. We have adapted their approach to highlight software structural changes over time through an interactive view that uses animation. In this view of the data, the analyst interacts with a time slider and can see city suburbs growing and shrinking to quickly get an overview of when and where hierarchical structural changes took place.

In order to test and validate our ideas, we conducted an experiment to validate the efficiency and effectiveness of basic tasks involved in the understanding of software architecture evaluation with and without our evolving city metaphor. Further, we were also interested in the usefulness and acceptance of our IVA approach as compared to the traditional workflow of the analysts.

1.3.3 Visual Analysis of Software Architecture Relations

Current software analysis tools tend to incorporate visualizations that focus more on the architectural structure as compared to the inter-dependencies of these software structures. Most tools depict these inter-dependencies using still images and provide the user with no direct means of examining their evolution. Further, they typically connect elements with straight line segments that lead to cluttered diagrams that are difficult to understand.

In order to address these issues, we extended our eCITY tool to incorporate the architectural ties between architectural structures. To achieve this goal, we represent relationships between architectural structures as Hierarchical Edge Bundles (HEBs) [5] on top of our evolving software city metaphor and unravel the evolution of these relationships through the use of animation. Additionally, we propose interactive particle animations instead of curvature, arrows, or colors to depict the direction of these relations.

Overall, our approach has been well received by visualization, Human Computer Interaction (HCI), and software engineering (software analysts) experts. In particular, the bundling of relations using the streets' space was found to be "interesting" and "highly useful", as it produces a compact visualization that minimizes clutter and contains more information. Additionally, we also present the insights gained in examining a real software system using our approach.

1.3.4 Visual Analysis in a Collaborative Environment

The comprehension of software is often a social activity involving personnel across many disciplines (i.e., architects, developers, engineers, etc.). Therefore, we introduce a collaborative framework that extends the principle idea of IVA to a scalable large high-resolution Tiled-Wall display.

In particular, we present a lightweight dispatcher framework to facilitate input management, focus management, and the execution of several interrelated yet independent visualizations. The main idea of this approach is to "brush and link" different visualizations or perspectives of the data via messages that are passed through our framework. Two case studies demonstrate that the proposed approach is indeed applicable in the context of collaborative software understanding tasks.

1.4 Organization of this Thesis

The remaining parts of this thesis are organized as follows: Chapter 2 provides background and surveys the state of the art in related fields for analyzing and understanding software artifacts. Chapter 3 describes a graph-based data model capable of representing software analysis data sets effectively and efficiently. This chapter also introduces a workflow-based approach that combines the generation of software measurements and visualizations. Further, it provides details on how these ideas are integrated and evaluated using our VIMETRIK prototype tool. In Chapter 4, we introduce an IVA approach integrated into a conventional analysis tool to examine the evolution of software architecture. Further, implementation details of our eCITY tool are provided and experimental results are discussed. Chapter 5 addresses the visual analysis of software architecture relations. This chapter discusses the mapping of architectural ties on top of existing visualization, the use of particle animation to depict direction, and the use of animation to depict their evolution. Chapter 6 applies our ideas in a collaborative environment. Finally, Chapter 7 provides an outlook and closing remarks.

2

Background and Related Work

> " *All truths are easy to understand once they are discovered; the point is to discover them* "
>
> – Galileo Galilei [1]

The work presented in this thesis leverages IVA to capture, process, and analyze information from various kinds of software artifacts. The basic idea of IVA is to generate knowledge from large and complex data sets through the combination of data analysis methods and interactive visualizations [62–64]. In the context of IVA, Coordinated Multiple Views (CMV) with linking and brushing play a pivotal role by enabling information drill-down processes [65]. Further, recent initiatives have focused on the synergy of visual and computational data analysis methods to form the field of Visual Analytics (VA). While IVA mainly focuses on the integration of data mining and data analysis into visualizations, VA aims to encompass other methods of analytical reasoning (i.e., machine learning, pattern extraction, cognitive and perceptual science, decision science, etc.).

In this chapter we survey related work with respect to visual analytics of engineering data, software architecture visualization, and CMV. We admit that the list of related work we review is by no means extensive. There are several excellent books and surveys on: 1) combining visual data analysis with interactive visualizations by Simoff et al. [66], Oliveira and Levkowitz [67], Keim [62], and Keim

[1] Was an Italian physicist, mathematician, engineer, astronomer, and philosopher who played a major role in the scientific revolution

et al [38], 2) a general overview of software visualization [43, 68], and 3) the state of the art of coordinated multiple views [54]. These materials contain more in-depth reviews of the respective fields. We try to extract and present only the most relevant aspects with respect to the contribution of this thesis. The interested reader is also referred to a book [40] by the European visual analytics consortium[2] that additionally discusses VA aspects such as data management, space and time analysis, cognition and perception considerations, and evaluation.

This Chapter is organized as follows. In Section 2.1, we discuss important concepts such as the combination of computational analysis methods and interactive visualization. Then, in Section 2.2, we present the state of the art of software architecture visualization. Next, in Section 2.3 we examine coordinated multiple views. Finally, closing remarks are presented in Section 2.4.

2.1 Visual Analytics

VA amalgamates techniques from graphics, visualization, interaction, data analysis, and data mining to support reasoning and sense-making for complex problems solving in a variety of fields [1, 69]; i.e., engineering, finances, security, geosciences, etc. There are quite a few similarities between these fields and software understanding in terms of data (software data is multivariate, relational, large, and abstract), reasoning (making sense of data), and tools (combination of analysis and visualization).

In general, VA consists of a pipeline of activities (see Figure 2.1) that refine and enrich the underlying data with semantics in order to understand it better [1]. Initially, in the *data foraging* loop, data is searched and filtered to extract elements of interest. This is predominantly a data mining step that consists of extracting all modules and module dependencies in a software code base. Next, in the *sense-making* loop, a hypothesis is formed, a data schema is structured to reflect the hypothesis, and the data is fit into it to validate or invalidate the hypothesis. This sense-making step can assist analysts in identifying a number of critical aspects; such as, modularity problems of a software system, non-compliance of coding standards, performance issues, etc.

[2] VisMaster (http://www.visual-analytics.eu/)

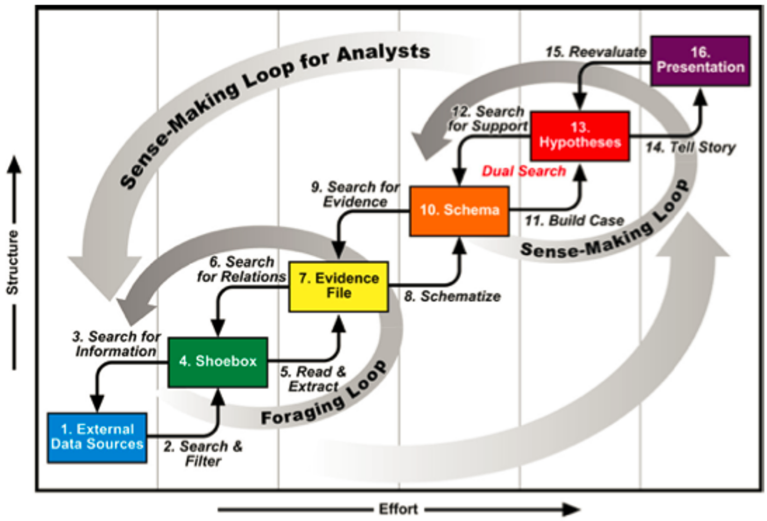


Figure 2.1. The Visual Analytics sense-making process [1]

However, while applying the VA pipeline to software analysis, there exists a large gap between the foraging and sense-making parts. In terms of the former, there are several software fact mining and metrics computation tools. While in terms of sense-making, there are are several interactive software visualizations. However, in practice they are separated into different tools. In this thesis, we investigate combining the two via an IVA means. Additionally, there are even some on-going research efforts to bridge this gap [14, 47].

In the forthcoming subsections, we examine related work with respect to the activities involved in the VA process – namely *data foraging* that consists of fact extraction and data mining as well as visual means of approaching sense-making. Since this is a fairly wide topic, we shall focus on introducing these topics and providing appropriate resources for more details. Additionally, we shall examine the applicability of VA to software analysis and present the most prevalent tools.

2.1.1 Data Foraging Loop

Generally, the *data foraging* loop can be further subdivided into fact extraction and data mining. Fact extraction covers the gathering of information from source code, binaries, and Source Control Management (SCM) systems such as CVS, Subversion, or Git. There are a number of well-known static analyzers that may either trade fact completeness and accuracy for speed and simplicity or perform full syntactic and semantic analysis at higher cost. Some well-known static analyzers include ROSE [70], Columbus [71], Eclipse JDT[3], and Recoder[4]. In our work, we wanted to develop on top of a well-known IDE, therefore, we choose to perform a full syntactical and semantic analysis using the Eclipse JDT.

Once data is generated through the fact extraction process, data mining techniques are applied to extract modules and their interdependencies [31]. However, extracting facts of large software systems results in a huge quantity of data; making it difficult to simultaneously keep all the data in memory and to discover patterns and relationships. The most common solution in practice is to store data pertaining to software facts in a relational database [72] and to query the database for knowledge discovery.

While the relational database approach has worked in resolving memory issues, it is severely hindered in recent times due to the exponential growth of the volume of data generated by users, systems and sensors, and the constraints of scalability over several servers. Once the data is modeled in this traditional sense, it is normally queried via Structured Query Language (SQL) to generate software metrics or measurements; such as complexity measures (i.e., McCabe, Halstead, etc), cohesion and coupling metrics [30, 31], or even encompassing source code analysis rules [47].

In addition, relationship database systems can often experience "implementation caused" problems due to any combination of the following reasons: if the data contains many relationships and requires joins of large tables, schema evolution over time, semi structured data,

[3] Eclipse Java Development Tools (JDT) (http://www.eclipse.org/jdt/overview.php/)

[4] Recoder Java Analyzer (http://sourceforge.net/projects/recoder/)

and scalability[5]. Furthermore, a change in the schema of the data sources requires major restructuring of the global schema through operations that are costly, complicated, and may not be performed automatically.

Major internet companies, such as Google, Amazon, and Facebook, had similar challenges in dealing with vast quantities of data that conventional relationship databases could not handle, therefore, they have already turned to the Not Only SQL (NoSQL) movement [73]. Although the original intention of this paradigm shift was towards modern web-scale databases, its schema-free approach and its independence from SQL as its query language has made it a viable option for many non-web based applications.

Typically, NoSQL databases are classified according to different data models. They are normally classified into five categories: key-value data stores, column stores, document stores, object-oriented databases, and graph databases. The interested reader may refer to the paper of Nayak et al. [74] for a detailed description of NoSQL data models, the types of NoSQL data stores, and the characteristics and feature of each data store.

In our work, we choose a graph database due to the inherent graph like structure of object-oriented software. In the graph database paradigm, the underlying data model consists of nodes and edges, where nodes represent software modules and edges depict dependencies between the software modules. A key feature of graph databases that we look to exploit is that they focus on finding relationships within massive amounts of data at the fastest possible speed. These features make the graph database approach an ideal replacement for a relational database. In the past, this adaptation might have been constrained due to the unavailability of high-performance graph database implementations. However, several promising projects have been developed in recent years, such as: Neo4j[6], AllegroGraph[7], and HypergraphDB[8]. Further, we have also seen developers and researchers apply this approach in the context of software engineering [75, 76].

[5] Graph Databases, NOSQL and Neo4j (http://www.infoq.com/articles/graph-nosql-neo4j/)
[6] Neo4j - The World's Leading Graph Database (http://neo4j.org/)
[7] AllegroGraph (http://franz.com/agraph/allegrograph/)
[8] HyhperGraphDB (http://www.hypergraphdb.org/index)

We decided to implement our graph database using Neo4j due to its performance, reliability, and scalability, as well as it being ranked the world's leading graph database by database monitoring site DB-Engines[9].

2.1.2 Sense-Making Loop

In the sense-making loop, data needs to be restructured according to a hypothesis in order to validate or invalidate it. This process typically consists of formulating hypotheses, executing queries, and examining results.

In traditional approaches, relational databases are queried with textual SQL type queries and results in the form of tabular data are scrutinized. However, analysts not only require knowledge of the query language but they also have to look into the data as well as its ontology. Typically, these textual queries present several issues for the database user, such as the necessity to identify the database classes, attributes, and relationship structure before writing a query, and issues relating to semantic and syntactical errors [77].

The work of Cammarano et al. [78] observes that most data analysis user interfaces take either one of two approaches. While one approach focuses on simplifying the query specification, the other examines the results through visualization metaphors and techniques. While specifying or presenting the results of queries visually has been beneficial in the context of exploratory search, it is the coupling of both these approaches that is of special interest [79]. In recent times, researchers have successfully applied a number of combined approaches to a wide variety of application domains, such as Network Security, Epidemiology, and Biomedicine [2, 80–82]. Figure 2.2 is an example of this approach that targets the detection and response of an infectious disease outbreak.

The above examples highlight the need to simplify query specification and the importance of innovative ways of viewing the results so that effective decisions may be made on large multidimensional measurement data – an aspect not so dissimilar to what software analysis tools strive for. In our work, we aim to bridge this gap through a Visual Programming Language (VPL) means that combines query specification and interactive software visualizations to present results

[9] DB-Engines Ranking of Graph DBMS (http://db-engines.com/en/ranking/graph+dbms/)

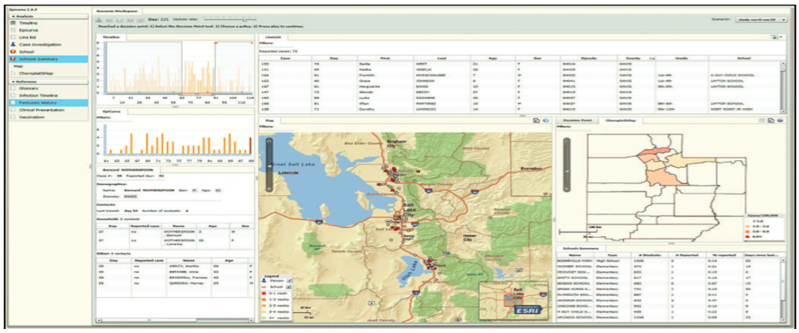


Figure 2.2. Epinome: a VA Workbench for Epidemiology data [2]

in a more meaningful manner. VPLs are languages that exploit visual representations in order to focus on the domain of interest instead of command languages, while interactive software visualizations encode software metrics in hierarchical representations of software systems.

Visual Query Specification

A number of techniques can be found as alternatives to command languages. A popular visualization for queries is the use of graph or network representations where nodes and edges represent components and their relationships. The Ecosystem Services Database [83] is a good illustration of this approach where users compare ecosystem service values across various geographic regions through the use of a graph-based visual query system.

Some researchers have focused on graphically representing the Boolean operations found in command languages. Representations of Venn diagrams [84] have been used to form graphical queries, where query terms are associated with a ring or circle and conjunction of terms are indicated using intersection of circles. Similarly, flow diagrams [85] have been used to depict conjunctions using sequential flows and dis-junction using parallel flows. Elmqvist el al. [86] present a visual canvas for constructing visual queries through the use of a graphical set representation that they refer to as DataRoses. Each DataRose comprises of a starplot of selected columns in a dataset that are displayed as a multivariate visualization that incorporates

dynamic query sliders into each axis. Tools such as InfoCrystal [87] and KMVQL [88] are similar in providing a means to find and select graphical representations of interest. The former employs iconographic representations while the latter makes use of Karnaugh maps.

Other interfaces found in literature are either based on the ubiquitous file-system browser interface [89] or focus on certain innuendos that assist the user in forming his query. Examples of the latter would be the work of Sinha and Karger [90] that assists the user through the use of navigation hints and the work of Trigoni [91] that lets the user improve a query over time by gradually revealing the underlying data.

Query Result Display

Traditional query interfaces display data items that meet query specifications in the form of tabular results. While this is still a useful approach, it often makes it difficult for the user to make correlations in large datasets. In the recent past, researchers have sought to tackle this problem by empowering users with better visual feedbacks.

Researchers such as Lucas el al. [92] and Mathew Ward [93] have implemented a variety of standard statistical charts as well as information visualization graphing techniques to communicate the results to the user. Similarly, systems such as Visionary [94] offer a direct-manipulation interface for browsing the results. The survey paper of Oliveira et al. [95] provides a closer look at such database visualization techniques.

In the context of software analysis, interactive software visualizations can be applied to present the extracted facts in a more meaningful manner than traditional methods. In Section 2.2, we provide a detailed overview of the relevant software visualization tools and techniques.

Coupling Queries with Results

The well known metaphor of a pivot table in a spreadsheet is used in the Polaris system [96] to incorporate both a novel query interface mechanism as well as an integrated visualization that displays correlations in data with respect to any attribute in the dataset. Similarly, the research of Livnat el al. [82] focuses on visual correlations of heterogeneous data to facilitate situational awareness and decision making processes. His work with Draper [81] introduces an interactive

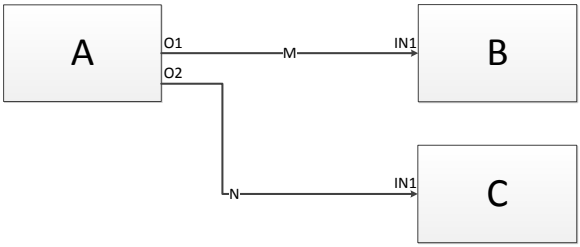


Figure 2.3. Simple flow-based diagram

radial query language for simplifying the tasks of searching for data correlation. They place icons representing individual entities around the circumference of a ring and allow the user to interactively focus on certain relationships by dragging relationship icons into the ring's interior.

In our work, we employ a diagram-based VPL that combines software related queries with query results. VPLs are languages that exploit visual representations in order to focus on the domain of interest instead of command languages. They facilitate users to program by manipulating or arranging graphical elements rather than writing textual source code. This gives users the ability to work with them at a higher abstraction level where they need no prior experience or knowledge to express their programming requirements. Thereby, providing end-users with a more intuitive way to create, modify, or extend parts of a software system.

Every VPL can be classified into one of three basic categories: icon-based, form-based, or diagram-based, depending on which type of visual expressions are used. In his thesis, Stehno [97] examines these categories in more detail. Our work is based on the concept of *boxes and arrows* which belongs to the category of diagram-based visual programming. In this flow-based paradigm [98], nodes can be thought of as "black boxes" and arrows as "arcs" that send data tokens to other connected nodes. Figure 2.3 shows the major entities of a flow-based diagram: A, B, and C are black boxes that process executing code components, and M and N are arcs that connect to their respective processes via the O1, O2, and two IN1 ports. Each node performs

a pre-described task as soon as it receives all the required tokens it needs for execution, while arcs carry numbers, arrays, or even pointers to objects as data tokens between a sending or receiving node. This principle is often referred to as the dataflow execution model.

2.1.3 Tools

Due to data being generated at an alarming rate, there is an increased interest and effort from both academia and industry towards VA solutions to assist in the sense-making of this data. On the one hand, there are a number of commercial vendors specialized in data discovery such as *Tableau*[10], *Qlik*[11], *TIBCO*[12], and the *Fraunhofer M-System* [99] as well as multinational corporations with Business Intelligence solutions such as *IBM*, *Microsoft*, *Oracle*, and *SAP*. While on the other, there are many open-source solutions that have come out of academia such as Gephi [100], GraphViz [101], Improvise [102], Protovis [103], R [104], and Infovis [105].

In general, research tools provide various state-of-the-art VA functionality that may include prototype techniques. However, the biggest challenge in using these open source toolkits is integrating them into an end-user application due to the lack of maintenance, development, and support. On the other hand, commercial tools are designed to be used as-is and are typically bundled with conservative visualization techniques.

For further details, the interested reader may refer to survey of Harger and Crossno [106] that compares existing open source VA solutions and the work of Zhang et al. [107] that examines current market trends. Additionally, relevant software visualization tools are listed in Section 2.2.3 and VPLs such as LabView, KNIME, OpenDX, Quartz Composer, and the Visualization Toolkit are covered in more detail in Stehno's thesis [97].

In our work, we extend KNIME [108] (see Figure 2.4), a visual data exploration and data mining tool, that is built as an Eclipse plugin. Using a flow-based approach, we enable software analysts to perform complex queries by configuring nodes that represent sub-queries and connecting them visually in a workflow editor. As shown in Figure 2.4, the query results can be examined through traditional views.

[10] Tableau Software (http://www.tableausoftware.com/)
[11] Qlik (http://www.qlik.com/)
[12] TIBCO Spotfire (http://spotfire.tibco.com/)

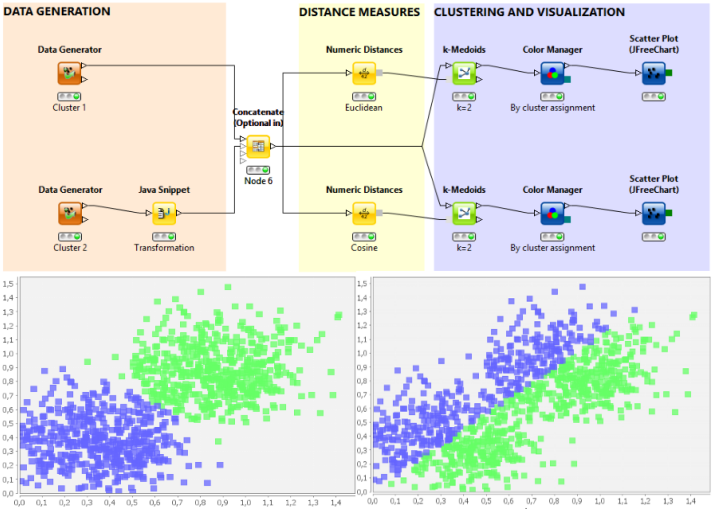


Figure 2.4. Sample KNIME workflow and views[13]

Additionally, the resulting dataflow networks can be analyzed through a custom *NetworkViewer* that embeds the results into interactive software visualizations. Thus, providing the analyst with a tool that can combine both computational analysis and software visualization.

2.2 Interactive Software Visualization

Software systems are an integral component of our everyday life, as we find them in tools and equipment all around us. It is imperative and at times critical to produce and maintain reliable systems, a well-designed and well-maintained architecture goes a long way in achieving this goal. However, due to the intangible and often complex nature of software architecture, this task can be quite complicated. The field of software visualization aims to ease this task by providing interactive tools and techniques to examine the hierarchy, relationship, evolution, and quality of architecture components.

[13] Picture taken from KNIME Blog (http://www.knime.org/category/blog/knime-blog-tech/). It shows a KNIME workflow with two scatter plots.

In this section, we present a discourse on the state of the art of interactive software visualization tools and techniques. Further, we highlight the importance of developing solutions tailored to meet the needs and requirements of the stakeholders involved in the analysis process.

2.2.1 Visualizing Architectures

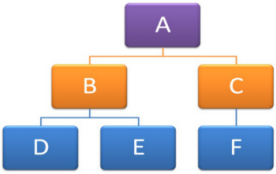
One of the core topics in the field of software visualization is a means to effectively visualize, navigate, and explore the software architecture of a system [44, 109, 110]. Generally, object-oriented software tends to be structured hierarchically - with packages containing sub-packages, which in turn contain classes that hold methods and attributes. It is this hierarchy and relationships between software components that is of interest when it comes to interactive software visualization [111].

In the context of visualizing software architectures, we explore representations of the global architecture of a system, such as tree, graph, and diagram model depictions. Further, we also investigate representations that highlight relationships between components as well as the importance of visualizing software metrics.

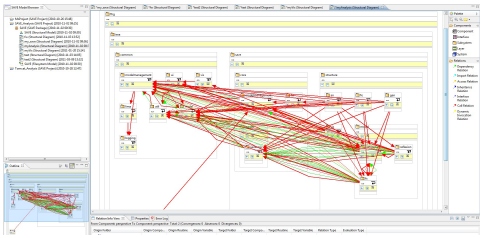
Architecture Representations

Tree structures are an ideal way of representing the hierarchical structure of software architecture. However, research in this area has shown the need to move forward from well-known techniques such as node-link layouts to more sophisticated ones to handle the larger hierarchies found in software systems nowadays [45]. Figure 2.5 shows both a generic node-link diagram as well as one found in a commercial tool. Inspection of these representations shows that they quickly become too large and utilize available screen space far too poorly for proper investigation. Further, the amount of textual information represented in the nodes as well as the way relationships are depicted should be revisited to avoid visual clutter and information overload [112].

In our research we have inspected several 2D visual representations [3] that may not be specific to just software visualization, but have been effectively applied to highlight the hierarchical structure of a software system [45, 113]. Here, it is important to note that a lot of these representations have been extended to 3D visualizations



(a) General node-link diagram

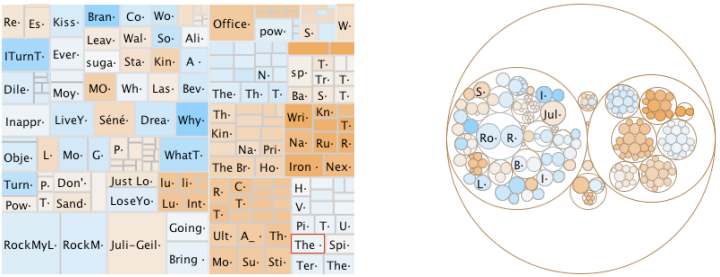


(b) As found in SAVE [4]

Figure 2.5. General and tool specific node-link diagram

[114–116]. While 3D approaches have been shown to display larger hierarchies and minimize clutter [117], they have also suffered from the well documented drawbacks of 3D visualizations, such as: object occlusion, cumbersome view adjustments, performance issues, as well as poor readability of 3D texts [118, 119]. Due to these drawbacks and the requirements of our stakeholders, we focus mostly on 2D representations.

The *Treemap visualization* (see Figure 2.6a), first introduced by Johnson and Schneiderman [120], is an effective means to visualize an entire software hierarchy. It is essentially a space-filling technique that displays hierarchical data as a set of nested rectangles. This is usually performed by a tiling algorithm that slices a box into smaller boxes for each level of the hierarchy, recursively, alternating between horizontal and vertical slices. “The resulting visualization displays all the elements of the hierarchy, while the paths to these elements are implicitly encoded by the Treemap nesting” [111]. In the context of interactive software visualization, Treemaps are used to represent methods as elementary boxes and classes as composed boxes. Several modifications of Treemaps appear in literature and in practice - some improve readability by enforcing an aspect ratio as close as possible to 1, while others have used irregular shapes such as Voronoi instead of rectangles to show more information [121]. Typically, designers are limited to the encoding of a single metric - the box color. While this provides a symbolic idea of how such a metric value is spread through the hierarchy, it is not simple to determine or represent metrics of enclosing entities [122].



(a) Rectangular TreeMap (b) Circular TreeMap

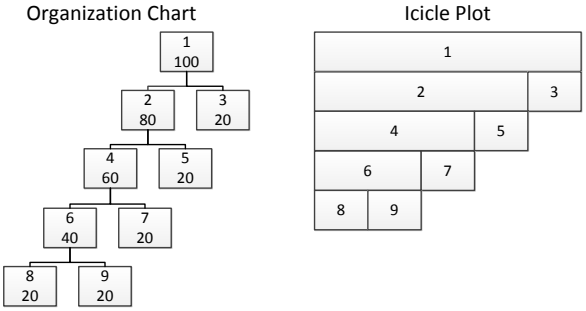
Figure 2.6. Rectangular and Circular TreeMaps using TreeViz[14]

Treemaps provide an extremely compact layout, however, they are limited by mainly showing the leaves of the software structure. Similarly, the *circular Treemap visualization* (see Figure 2.6b) and variations of it have been researched in order to have circles fill the available space more efficiently [123].

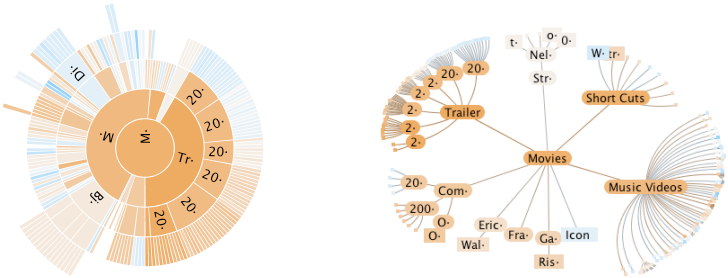
The *Icicle Plot* principle of Figure 2.7a is where a line represents a tree level and each line is split according to its number of children [3]. While Icicle Plots provide better understanding of structural relationships as packages can be used as root and classes and methods as tree elements, scalability and navigation may be an issue with hierarchies of large systems [122]. Typically, two metrics maybe encoded in the visual representations: node size and color.

An alternative space-filling technique to nested geometry is the use of a *Sunburst visualization* that focuses on adjacencies instead [124]. This technique was first proposed by Stasko and Zhang [125], where they utilized a circular or radial display to depict the hierarchy rather than a rectangular layout (see Figure 2.7b). In a sunburst, the hierarchy is laid out radially with the root at the center and discs or portions of discs as deeper levels further away from this center [126]. In contrast to the Treemap techniques mentioned earlier and similar to the Icicle Plot, designers have the added flexibility to encode two distinct metrics: the angle swept out by an item and its color [122]. Studies have shown the performance of localization, comparison, and identification tasks in Treemap and Sunburst visualizations to be

[14] TreeViz (http://www.randelshofer.ch/treeviz/)



(a) Icicle Plot



(b) Sunburst TreeMap (c) Hyperbolic TreeMap

Figure 2.7. Icicle [3], Sunburst, and Hyperbolic layouts using TreeViz[14]

comparable, however the Sunburst is found to be easier to learn and more pleasant [127]. While screen-space is better utilized as compared to node-link diagrams, scalability and navigation may still be an issue in larger systems.

Another approach is to make use of the hyperbolic space, which intrinsically provides more space than a layout that employs Euclidean coordinates. This well-established technique is more commonly referred to as the *hyperbolic tree layout* (see Figure 2.7c) and was first introduced in the context of information visualization by Lamping et al. [128]. Essentially, it lays out the hierarchy in a uniform manner on a hyperbolic plane and maps this plane on a circular display region.

A key feature of this technique is referred to as *fisheye distortion*[15], where components tend to diminish in size as they move outwards. This leads to a larger representation of the center or focused area while still displaying the overall structure of the tree. Hyperbolic trees provide an interactive overview of a hierarchy; they show both detail and context at once. Initially the root of the hierarchy is placed in the center, however, the display can be transformed to bring another node into focus through interaction. It would probably be best to encode metrics through the use of color alone, as varying the node size would adversely affect the layout algorithm. When the graph is deemed too large to be rendered effectively, nodes are pruned together and may be interactively expanded to reveal the subtree structure.

Visualizing Relationships

In contrast to visualizing the software hierarchy of a system, visualizing relationships of the software system is a more complex task. This is due to both the higher amount and the different types of relations that exist in a system, such as: inheritance, method calls, dynamic invocation, accesses, etc.

Generally, *graphs* have all the characteristics required to represent relationships of a software system. This is typically done by expressing software components as nodes and relationships between them as edges [129]. However, this often leads to the visualization of an extremely large graph due to the high inter-connectivity between the large amount of components found in software systems nowadays. Thus, the resulting visualization tends to be extremely confusing and cluttered - it becomes difficult to discern between nodes and edges due to the cluttering, overlapping, and occlusion of edges (see Figure 2.8).

A well-known approach to remedy this clutter issue is to replace node-link diagrams with a square matrix that has matching row and column labels. The matrix then highlights the number of relations between row and column elements within each matrix entry, possibly through some visual representation [130]. This well-known technique is often referred to as the *Dependency Structure Matrix* [131] in literature and provides a compact and uncomplicated representation of relations in a complex system. However, keeping a mental map of the system hierarchy can still be an issue in these visualizations.

[15] A Brief Tour of Nonlinear Magnification (http://alan.keahey.org/research/nlm/nlmTour.html/)

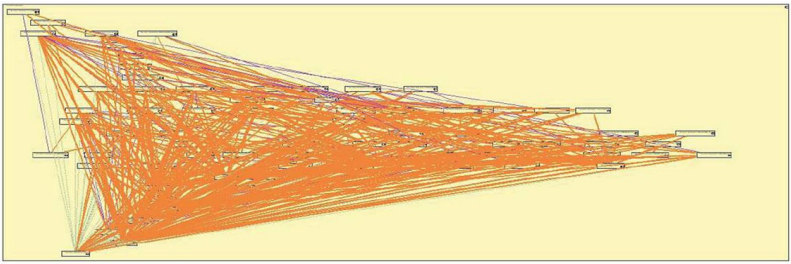
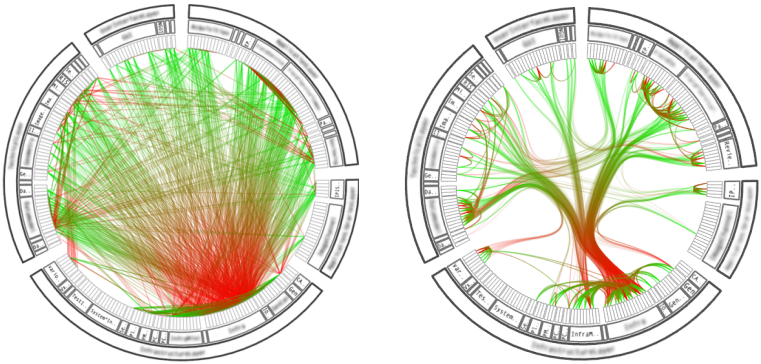


Figure 2.8. Cluttered software architecture [4]

The most accepted graph-based software visualization in the field of object-oriented software engineering are *Unified Modeling Language (UML) class diagrams*. This modeling language was created and developed by the Objected Management Group and has since become the industry standard for modeling software systems[16]. Its main purpose is to portray inter-class relations, such as: composition, inheritance, generalizations, aggregations, and associations. However, due to the amount of textual information depicted by each component such as the listing of methods and variables, these graphs grow exponentially with each additional component or class notation and are highly prone to information overload. Some researchers have looked at reducing the visual complexity associated with such graphs by reducing the number of overlapping edges, the use of orthogonal layouts, the horizontal writing of the labels, and edge bundling [132–134]. While some success in reducing the complexity has been achieved, the drawbacks associated with node-link diagrams such as poor screen-space management and information overload still need to be tackled.

More recently, researchers have experimented with different layout and filter techniques in order to resolve the clutter issue. An example of this is the work of Pinzger et al. [135] that focuses on the creation of condensed and aesthetically pleasing graphs that show information relevant to solve a given program comprehension task. Their solution was to use nested graphs and a feature that allowed to add and filter appropriate nodes and edges. Other researchers such as Holten [5] have chosen to implement better space-filling techniques in combination with improved edge representations. Holten's approach was to place software elements on concentric circles according to their depth in

[16] UML FAQ (http://www.uml-forum.com/FAQ.htm)



(a) $\beta = 0$ (b) $\beta = 0.75$

Figure 2.9. Hierarchical Edge Bundles (HEBs)[5]

the hierarchical tree and then to display edges above the hierarchical visualization (see Figure 2.9). Further, he extended the work of Fekete et al. [136] that used spline edges to replace explicit arrow directions, in order to reduce the visual clutter and edge congestion by allowing edges to bundle together according to a parameter (see Figures 2.9a and 2.9b). Similarly, techniques displaying, clustering, and filtering edges on top of structural representations can be utilized in other visualizations (i.e., Treemaps, circular trees, etc.) to represent the hierarchical graph structure of a software system.

Another approach to resolve the issues of cluttered 2D graphs is the use of 3D visualizations [137], where the user can access a view without occlusions. However, 3D representations of large graphs have their own problems, such as: navigation can not only be difficult but also disorienting [138], object occlusion, performance issues, and text illegibility [118]. For the purpose of completion it would be prudent to mention some of the more prominent work in the area of 3D interactive software visualization. Some researchers in this field have experimented with real-world metaphors to take advantage of the intuitiveness of these representations [139]. For example, the *City* or *Cities* metaphors are often used to depict relationships through a visually understandable metaphor [114, 140], where cities (packages) are connected via streets (two-directional calls) and water (uni-directional calls). Similarly, researchers have realized the *Solar System* [141], *Island* [140], and *Landscape* [115, 142] metaphors, where the respec-

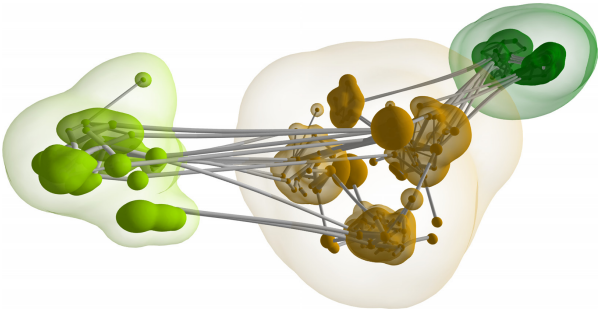


Figure 2.10. Clustered graph layout [6]

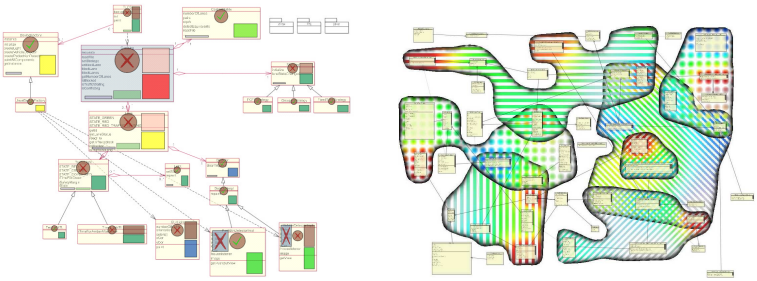
tive relationships between each contributing element is exploited to depict packages, classes, and their relationships. Another interesting approach towards handling large and complex graphs is the *clustered graph layout* (see Figure 2.10), where clustering, dynamic transparency, and edge bundling are used to visualize a graph without altering its structure or layout [6].

Visualizing Software Metrics

The incorporation of software metrics is an important component in the analysis of a software systems architecture, as they not only provide an insight into the quality of the software design [143, 144] but also a means to monitor this quality throughout the design process [145]. Typical static software metrics express different aspects of a complex system, such as: design complexity, resource usage, and system stability.

The idea behind metric-centered visualizations is to transform numerical statistical data into a visual representation that is easier to understand and grasped far more intuitively and instantaneously [146]. Here, the greatest challenge is to find an effective mapping from a numerical representation to a graphical one that enhances the structural visualization [147].

In this section, select visualization techniques that implement static software metrics are highlighted - the purpose of which is to provide an idea of the implemented approaches. One such approach is to combine them with UML class diagrams. An example of this is the *MetricView* (see Figure 2.11a) visualization that displays metric icons on top of UML diagram elements [7].



(a) Metric View (b) Areas of Interest

Figure 2.11. Metric View [7] and Areas of Interest visualizations [8]

An extension of this approach is the *areas of interest* (see Figure 2.11b) technique developed by Byelas and Telea [8]. They apply a layout algorithm that groups software entities with common properties, encloses these entities with a contour, and adds colors to depict software metrics. In order to distinguish overlapping areas, each area is given its own texture, such as: horizontal lines, vertical lines, diagonal lines, and circles. Further, shading and transparency techniques are used to improve the distinction between several areas.

In visual representations other than UML Diagrams, similar approaches have to be implemented in order to combine metrics and structural information. An example of this is the work of Holten et al., where they used texture and color to show two different software metrics on a Treemap [148]. Their results show that the combination of color and texture provides high information density, assists in finding correlations between metrics, and can reveal patterns and potential problem areas.

To visualize multiple aspects of a software system, Lanza et. al introduced the concept of *polymetric views*, where the visualization of a software is enriched with software metrics [149]. Essentially, they propose a node representation that encodes upto five distinct metrics; node width, height, x and y-coordinates and color, and edge width and color. They applied this to an inheritance tree where nodes represent classes and edges depict the inheritance relationship between them. Node width and height is used to encode the number of attributes and the number of methods. Further, a color tone is applied to represent the number of lines of code.

Other solutions found in literature that incorporate software metrics include 3D visualizations and filter techniques. In 3D visualizations, the encompassing visual entities have been encoded with software metrics [141, 150]. Another technique that may be applied in the analysis of system metrics is the use of filters, an example of this can be found in the Solar system metaphor, where filters may be applied to the overall system to visualize planets with metric values that lie within a chosen interval [141].

2.2.2 Visualizing Architecture Evolution

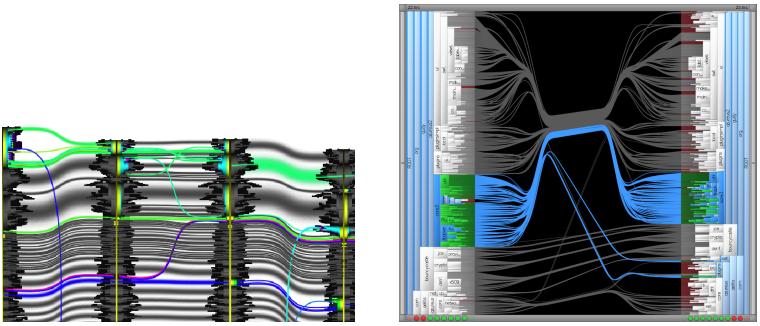
A general obstacle with regards to software evolution visualization is coping with the complexity that emerges from the huge quantity of evolution data; it is quite common to have hundreds of version of thousands of files [49]. The technical challenges associated with extrapolating this historical data are deemed out-of-context with respect to this thesis, instead, the focus will be on visualizing the evolution of the software architecture.

Real software solutions undergo continuous change to meet new requirements, adapt to new technology, and to repair errors [151]. Inevitably, the software in question magnifies in both size and complexity, often leading to a situation where the original design gradually decays unless proper maintenance is performed [50]. As such, visualizing the evolution of the software architecture is one of the key topics in the field of software evolution visualization [111]. It is essential to have a global overview of the entire system evolution in order to explain and document how a system has evolved to its present state and to predict its future development [46].

In the context of visualizing software architecture evolution, we first focus on how the global architecture of the software changes with each release and then examine at how relationships and metrics evolve within each version.

Visualizing Hierarchical Changes

Since software maintenance is performed mainly at code level, most visualizations have implemented a 2D line-based approach to represent the software evolution [152, 154, 155]. Generally, the adopted approach is to visually map a code line to pixel line, where color is typically used to show the age of a code fragment [154]. Additional focus has been to



(a) Code Flow [152]

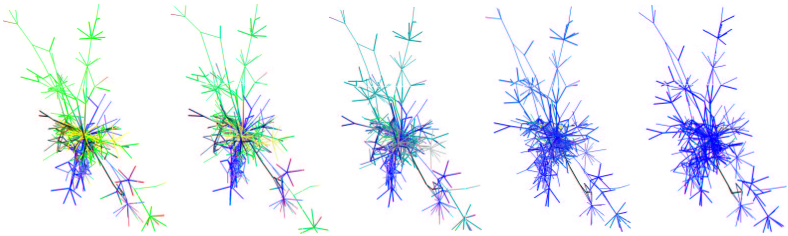
(b) Visual comparison of hierarchically organized data [153]

Figure 2.12. Visual comparison of two source code versions

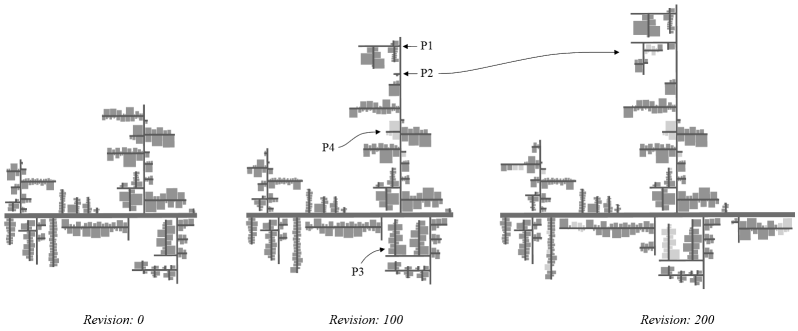
develop interaction techniques that allow users to effectively navigate and explore the data [155]. In order to highlight the state of the art in this traditional approach, the *Code flows* visualization technique [152] is briefly examined. Figure 2.12a shows an evolution from left to right of four versions of a source code class. This technique employs an icicle layout and bundled edges to show how a source code line changes over subsequent versions. Source code lines that do not change from one version to another are colored black, while code lines that changed are highlighted using different colors. In general, these tools are successful in tracking the line-based structure of software systems and reveal change dependencies at given moments in time [155]. However, they lack the sophistication to provide insight into attribute changes and more so the structural changes made throughout the development process.

In contrast, there are only a few visualizations aimed at representing structural changes of a system architecture over time [111]. As explained earlier, there definitely exists a requirement to monitor the evolution of a systems architecture, however, current graph animation algorithms are limited and need to mature further to handle this requirement [68].

One such approach, is the work of Holten et al. that presents a technique aimed at comparing the software hierarchies of two software versions [153]. To better compare the two versions, the algorithm tries to position matching nodes opposite to each other. This technique is



(a) Inheritance graph of successive versions [156]



(b) Evolving city layout [51]

Figure 2.13. Graph and City based layouts for software evolution

presented in Figure 2.12b, where the source code of Azureus v2.2 is displayed on the left and v2.3 is portrayed on the right. Nodes that are present in one version but not the other are highlighted via red shading. Further, the Edge Bundles technique of Section 2.2.1 is used to highlight and track the selected hierarchy.

Collberg et al. describe a system that visualizes the evolution of a software system using a graph drawing technique that handles a temporal component for the visualization of large graphs [156]. They accomplish this by utilizing a force-directed layout to plot call graphs, control-flow graphs, and inheritance graphs of Java programs. Changes that the graphs have gone through since inception are highlighted through the use of color. Nodes and Edges are initially given the color assigned to its author (red, yellow, or green) and progressively age to blue (see Figure 2.13a).

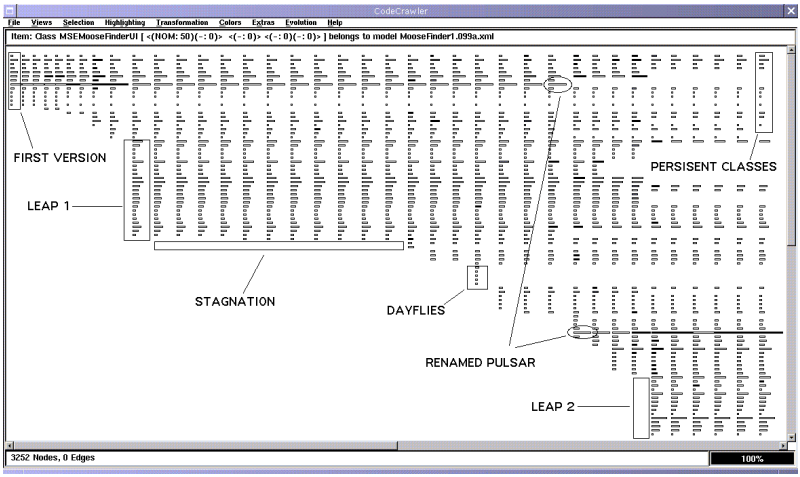
Lately, there has been some effort by researchers to extend known metaphors to handle the evolution of software systems. Steinbrückner et al. have an interesting approach that implements the city metaphor for the representation of large software systems in the form of evolving software cities [51]. Their work is illustrated in Figure 2.13b, where a system grows from an initial 389 classes to 439 classes in revision 100 and 466 classes in revision 200. In this implementation of the city metaphor, streets represent Java packages and building plots represent Java classes. The sequence of visual depictions aims to highlight basic changes in the software structure, how elements maybe added, removed, and moved within the software hierarchy. Further, they extend this general representation to address the needs of two distinct application scenarios by: 1) applying an *evolution map* that uses contour lines to show different versions of each subsystem and 2) using a *modification history map* that uses a contour line map combined with property towers that depicts the number of modifications as height and modification date as color.

Visualizing Software Metrics Evolution

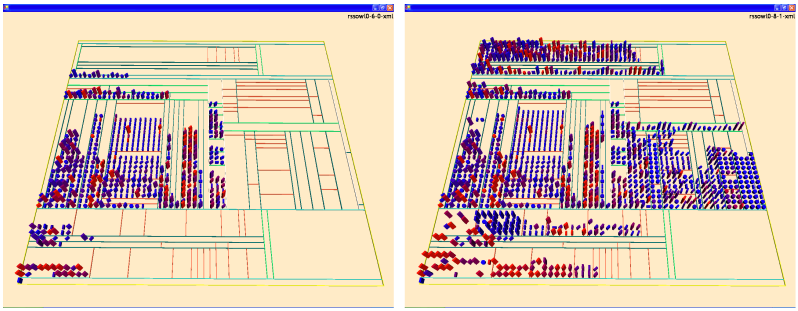
As covered in Section 2.2.1, visualizing relationships is an extremely complex task that is further compounded in the case of software evolution. Typically, researchers and practitioners focus more on the logical coupling between source code artifacts, as it can be encoded easily into metric values [157].

Software metrics are an ideal abstraction as they encapsulate, summarize, and provide essential quality information about source code [10]. As such, they are essential in providing a continual understanding and analysis of the quality of a system during all phases of the product life cycle. Instead of tedious, inefficient, and hard to grasp numerical representations, metrics tend to be mapped to graphical characteristics so that they may be intuitively interpreted. In this section, we explore the state of the art in the visualization of software metrics across different software versions.

The *Evolution Matrix* is a visualization technique that provides an exploratory view of an object-oriented systems evolution, both at the system and class granularity levels [9]. In this work, Lanza et al. combine software visualization and software metrics by using two-dimensional boxes to represent classes and encoding metric measurement of the classes to the width and height of the boxes. In the



(a) Evolution Matrix



(b) Two frames of RSSSowl using VERSO

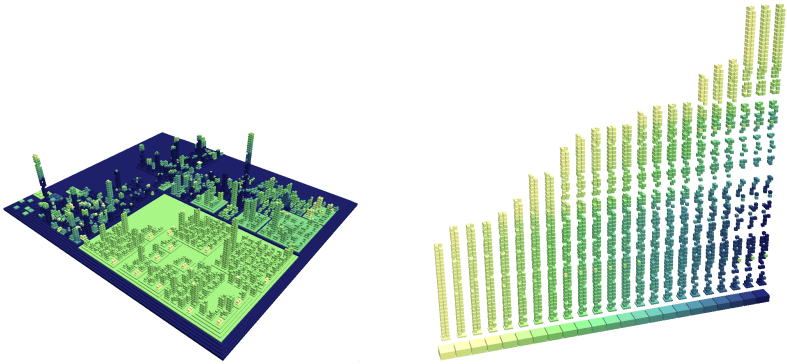
Figure 2.14. The Evolution Matrix [9] and VERSO [10] tools

example of Figure 2.14a, they use the metric *number of methods* for the width and *number of instance variables* for the height, columns to represent different versions of the software, and rows to depict different versions of the same class. At the system level, this technique recovered the following characteristics regarding the evolution of a system: size of the system, addition and removal of classes, and growth and stagnation phases in the evolution. While at the class level, it shows if the class grows, shrinks, or stays the same from one version

to another. These features allows the expert to analyze a number of interesting aspects, such as a class growing and shrinking repeatedly, a class suddenly exploding in size, or a class that had a certain size but lost its functionality.

The visualization framework by Langelier et al. also facilitates the analysis of software over many versions [10], albeit in a slightly different manner. Instead of employing a technique that displays the entire system evolution in one picture [9], they rely on animated transitions from one version to another. As Figure 2.14b shows, there are different static representations for each subsequent version; the image on the left is a previous version and the image on right is the next. The user controls forward and backward navigation in time, which in turn animates three graphical characteristics that are mapped to metric values - color, height, and twist. While the animations are of a short duration, they are well-designed and help attract the attention of the viewer towards program modifications [10]. This work of Langelier et al. contains references to extensive case studies aimed at detecting both evolution patterns and known anomalies. With respect to evolution patterns, users were able to identify constantly growing classes, quick birth and death of classes, and explosions in complexity in a short time-span. On the other hand, while looking for common anomalies, patterns such as the *God Class* or *Shotgun Surgery* were observed. The former is detected when a class constantly grows in complexity and coupling, while the latter occurs when a class constantly grows in terms of coupling and whose complexity increases globally but with an up-and-down local pattern.

Wettel and Lanza present interactive 3D visualizations in their *CodeCity* tool that examines the structural evolution of large software systems at both a coarse-grained and a fine-grained level [11]. At a coarse-grained level of granularity, classes are shown as monolithic blocks that lack details of the internal structure. While at the fine-grained level, the focus is on methods that appear as building bricks. Figure 2.15a shows this fine-grained representation, where classes are illustrated as buildings located in districts that represent the packages in which the classes are defined. Metric values are then encoded in the visual properties of the city artifacts; class properties such as the number of methods and number of attributes are mapped on to the buildings' height and base size, package depth is mapped on the districts' color saturation. Further, the age distribution of classes is represented through an Age Map color mapping, where the color



(a) A fine-grained Age Map (b) Timeline of a single class

Figure 2.15. Fine-grained and coarse CodeCity visualizations [11]

scheme ranges from light-yellow for recent entities to dark blue for earlier versions. Similar to the work of Langelier et al., back and forth transitions through the history of the system allows the city to update itself and reflect the currently displayed version. Additionally, at a finer level-of-detail the entire evolution of a single class or package may be tracked (see Figure 2.15b).

Pinzger et al. introduced a multivariate visualization technique that can display the evolution of numerous software metrics related to modules and relationships [12]. Figure 2.16 depicts this multivariate technique with data pertaining to 20 metrics, 7 modules, and 7 subsequent source code releases. In this approach, *graphs* and *Kiviat diagrams* are combined to graphically represent several metric values. The individual Kiviat diagrams present quantitative metrics, where low values are placed near the center of the Kiviat diagram and high values are found further away from the center. Dependency relationships between source code entities is highlighted by the layout of the diagram and the relationship between modules. Furthermore, this approach encodes the temporal aspects of multiple versions through a rainbow color gradient, where different colors indicate the time period between subsequent releases. Finally, the amount of coupling between two modules is represented by the width of edges connecting Kiviat diagrams. While, this visualization contains lots of information and can help identify critical source code entities or critical couplings, it

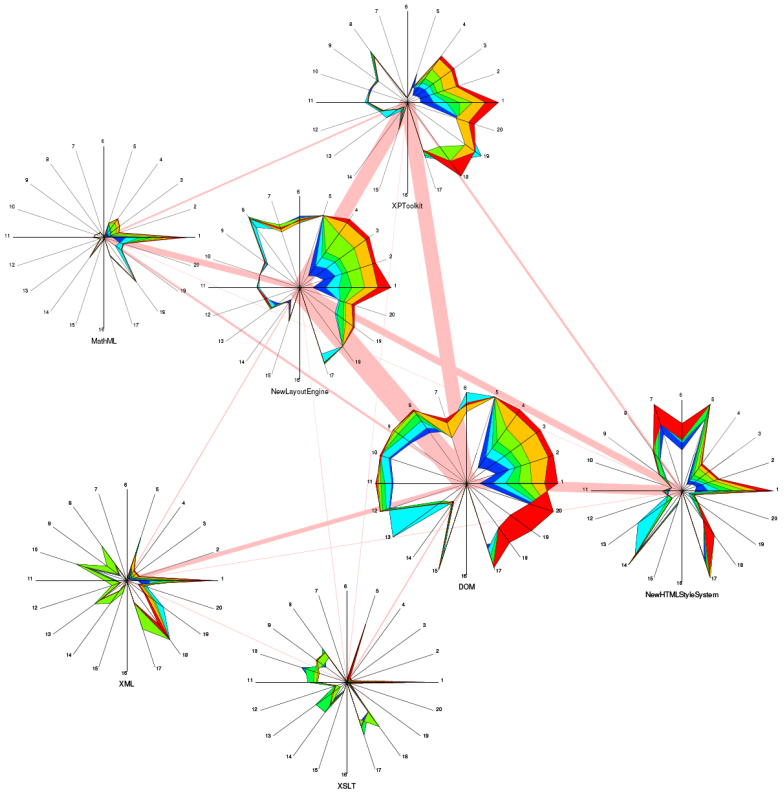


Figure 2.16. Multivariate visualization using Kiviat diagrams [12]

requires a good knowledge of software metrics. A positive feature of this technique is that all information regarding metrics and evolution is represented in a single static view that requires no animation. However, at times the color stripes overlap making it futile to discern the corresponding metric values. This problem of overlapping has been solved using 3D Kiviat diagrams that displays each version of the software on a different level of elevation [158].

2.2.3 Tools

There are a number of tools available both in academia and industry that cater to the various needs of stakeholders. On the one side, vendors have developed commercial Architecture Visualization Tools (AVTs): Lattix, Enterprise Architect, NDepend, Klockwork Architect, IBM Rational Architect, Bauhaus [45], etc. While on the other, the research community has also produced numerous tools: SHriMP [159], BugCrawler [160], DiffArchViz [161], etc. Commercial tools are generally designed to be used as-is, while research tools are open-source that allow users to customize them.

The main aim of these tools is to employ a combination of metaphors and techniques presented in this paper to assist technical users, project managers, and researchers in analyzing software architectures. The study of Telea et al. [45] shows that the mainstream masses are starting to realize the potential of these visualization techniques. For example, tools such as Lattix and NDepend have incorporated newer diagram-layout techniques, realizing the limitations of traditional node-link diagrams. However, this modernization of AVTs is much slower than the advent of cutting-edge visualization solutions.

AVTs typically support a combination of the following tasks: "comparing desired and actual architectures, identifying architecture violations, highlighting architecture patterns or layers extracted from code bases, assessing architecture quality, and discovering evolutionary patterns such as architectural erosion" [45]. However, no single tool can satisfy all these needs and requirements, as they differ in the features they provide, the audience they cater to, and the tasks they support [162].

The reader may refer to the work of Babu et al. [163] for a thorough comparison of AVTs according to the taxonomies they support. A closer inspection of these taxonomies is required, as it is imperative that visualizations are constructed to address problems and issues faced by the users of the system, rather than just provide 'pretty pictures'. The challenge often is that different stakeholders, such as: architects, developers, maintainers, and managers, require contrasting tools and techniques to delve into different levels of details. In the context of software architecture, several researchers, such as: McNair et al. [162] and Panas et al. [140], have conducted in-depth analysis of what to visualize and how best to achieve it. A good synopsis of these findings can be found in the survey of Ghanam et al. [44].

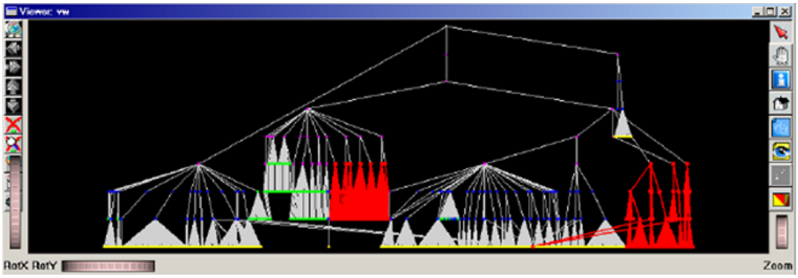
The most significant lesson learnt from the above-mentioned surveys is not to lose sight of the audience and to conduct appropriate evaluations where possible to determine the true worth of a proposed interactive software visualization; i.e., does it allow for a more thorough analysis (number of issues detected), does it perform tasks more efficiently (task completion time), etc.

2.3 Coordinated Multiple Views

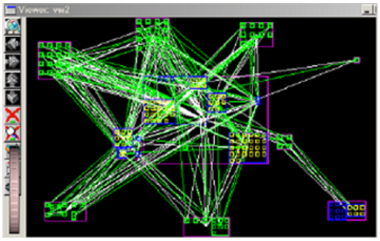
In this section, we provide a brief overview of CMV and highlight some implementations in the software visualization domain. For a more detailed review on the state of the art in CMV, the reader can refer to the well-known survey paper of Roberts [54].

There exists no single visual representation that can display all the relevant aspects of a complex dataset. IVA systems tackle this issue by combining different views of the same data in a way that the user can make relevant correlations or disparities. This is typically accomplished by simultaneously displaying, exploring, and analyzing different data variates in multiple side-by-side linked views. These views may include scatterplots [164], matrices, parallel coordinates [165], histograms, or even specialized domain-specific views (e.g., a combined visualization of software structure and metrics [166]).

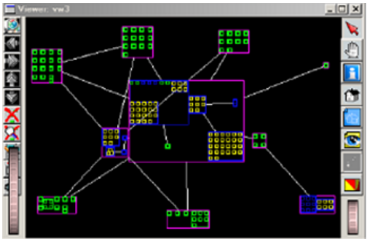
There are a lot of well-known visualization systems based on the CMV approach; such as Improvise [102], SimVis [167], Snap-together Visualization [168], XmdvTool [93], etc. These tools follow the basic principle highlighted by Baldonado et al. [53] – brush (select) interesting data subsets in a visual display to instantaneously highlight related data items in the linked views. This brushing and linking approach enables a "divide and conquer" means to examine complex datasets, where each view concentrates on a subset or sub-dimension of the data. Essentially, CMV create a focus+context visualization [169] where the selection equates to focus and the other data views provide its context. Complex queries can then be defined via several brushes or logical combinations [170]. These combinations may be through a feature definition language [171] or in a conjunctive visual form [172].



(a) Hierarchical system decomposition



(b) Selected set-of-interest



(c) Filtered set-of-interest

Figure 2.17. CMV using Soft-Vision [13]

In literature, there already exist some implementations of CMV in the context of software visualization [13, 48]. The Soft-Vision [13] framework highlights the typical approach of allowing users to interactively explore software analysis datasets through interactive views that present the data at different levels of abstraction. Figure 2.17 depicts a typical use-case with three different views. In Figure 2.17a, the containment relations of a software system are shown using a tree visualization. Components selected in this view are brushed and linked into the second view (see Figure 2.17b). This second view uses a "boxes in boxes" nested layout to show containment and association relations. Displaying all this data leads to information overload, however, the user interactively filters out the "uninteresting" association relations to generate the third view (see Figure 2.17c).

2.4 Concluding Remarks

In this chapter, we presented the foundations of our work as well as a literary review on the most relevant fields. Our research shows that both the VA and interactive software visualization domains have evolved significantly in recent years giving developers new tools to better understand, evaluate, and develop software and help managers to monitor design and refactoring issues. However, there remains the need to incorporate these cutting-edge tools and techniques with standard software development and maintenance practices.

Current practices consist of analysis tools that extract facts from source code, tools that refine these into software metrics, and tools that present these facts using visualization techniques. However, as the study of Telea et al. [47] suggests there is still the need for a combined analysis and visualization approach. In our work, we bridge this gap using a NoSQL approach, a graph database back-end, and a workflow based front-end.

Ultimately, our goal is to assist software analysts who typically need to either adapt pre-existing software metrics or define their own custom metrics. Current tools and approaches for software analysis (e.g., [99], [173], [174], [175], etc.) provide different degrees of customization, however, they typically require not only an in-depth knowledge of the model and underlying semantics (e.g. naming conventions) but also the technical know-how of how to formulate textual queries (i.e., SQL, Smalltalk, etc.). Instead, our approach aims at providing easy and intuitive means for the customized analysis of a software system in terms of software measurements or metrics, such as computing cyclomatic complexity, analyzing dependencies or call traces, or using statistical analysis to find issues.

Closely related tools are: ConQAT [176], a software quality assessment toolkit that employs a pipes-and-filter architecture, Specula [177], a goal-oriented approach for the composition of Software Project Control Centers (SPCCs), Sextant [76], a tool for specifying and visualizing software metrics for Java source code, and WiggleIndexer[17], a tool that allows the indexing of Abstract Syntax Trees (ASTs) using a graph database. Although these tools have some similar features, they do not employ workflows to combine computational analysis and software visualization tools. Both ConQAT and Specula also allow users to connect “configurable units” and “control components”

[17] WiggleIndexer (https://github.com/raoulDoc/WiggleIndexer)

respectively; however, these "units" and "components" focus more on high-level analysis tasks and less on how to combine low-level software metrics to create these high-level measurements. In contrast, our workflow approach is centered around collecting, aggregating, and visualizing both low-level artifacts and high-level software measurements. Additionally, we focus more on CMV of the data that maybe connected at any point in the pipeline.

Similar to our approach, WiggleIndexer has an equivalent graph database model and querying mechanism; however, users need to have in-depth knowledge of both the model and the underlying Cypher querying language. On the other hand, although Sextant provides several visualization capabilities, it contains significantly fewer details about the system and the analysis needs to be performed in the Service Modeling Language (SML) instead of with workflows.

Additionally, while visualizing software structures some visualization techniques such as node-link diagrams are well-known in industry. Other techniques such as parallel coordinates and bundled diagram layouts are less-known. The software analysis community has not made widespread use of these recent advances. There is a definite need to bridge this gap, as software systems are getting far too large to be analyzed through traditional means alone. This delay in adopting new technology may be due to the stakeholders not having enough time to try out every new tool, lack of knowledge with respect to technical visualization terms often used in marketing these tools, or simply a reluctance to try unknown visualization metaphors and techniques [45].

In our work, we have looked at ways to bridge gaps in the VA and interactive software visualization domains. In the past four years, we have closely worked with software analysts at the Fraunhofer Institute for Experimental Software Engineering (IESE) to tailor tools according to their specific needs and requirements. As a result of these efforts, we have developed research prototypes such as VIMETRIK and eCITY that show a lot of promise in the above-mentioned domains.

3

Visual Analysis of Software Measurement Data

" *The goal is to turn data into information, and information into insight* "

– Carly Fiorina [1]

In recent times, multidimensional visual analysis is becoming more and more important especially in the area of software measurement and analysis. This is due to the fact that most of the data from software measurement is multivariate. Analyses of such measurements of software systems tend to result in a huge amount of multidimensional data, in some cases even approaching "big data" analytics. In this regards, standard software analysis tools are limited by their lack of ability to process huge collections of multidimensional data sets; current tools are designed to support well-known metrics and are limited due to their reliance on relational databases and fixed schemas. There are tools that facilitate the customization of software metrics, however, they are often quite difficult to use. The end-user requires extensive knowledge of the underlying data schema and the querying language.

Further, to gain an insight of a systems' quality, most software analysis tools provide traditional means to explore the generated data. There exists a huge gap between these tools and interactive software visualizations that aim to present extracted facts and measurements in a more meaningful manner than traditional methods.

[1] Former CEO of Hewlett-Packard

In this chapter, the above-mentioned shortcomings are addressed through an innovative means to facilitate the specification and visualization of user-defined software system measurements. The key ingredients of our methodology are a schema-less data access path to an underlying data model, a workflow-based approach to define metrics, and the ability to visually depict the results of these queries. In terms of the latter, we aim to provide end-users with both traditional views (i.e., tabular views, scatter plots, box plots, histograms, line charts, etc.) and interactive software visualizations. Additionally, we report on a live-data prototype of an interactive visual analysis tool called VIMETRIK to explore our ideas in the wild. This includes findings of a preliminary study that illustrates the intuitiveness and easy-to-use means of our approach to understand software measurement and analysis data.

This chapter is organized as follows. Section 3.1 provides context and motivation. Then, in Section 3.2, documents our approach of using graph databases and workflows. Further, we depict how our ideas can be implemented in a live-data prototype. In Sections 3.3 and 3.4 we report on the preliminary study setup and results. Finally, closing remarks are presented in Section 3.5.

3.1 Motivation

Software systems nowadays tend to be large, complex, and heterogeneous in nature and face increased pressure on delivery time and product quality. Studies estimate that up to 80% of the software costs occur in the maintenance phase, out of which 40% goes into understanding the software system [178]. In this context, various measurements or software metrics are often utilized to obtain objective, reproducible, and quantifiable measurements to assist developers in quality assurance testing, software debugging, and software performance optimization. As such, independent of the application area different measurements are scrutinized to ensure that the system in question performs optimally, is safe and reliable, or is of high quality.

Analyses of such measurements of software systems tend to result in the scrutiny of a large amount of analysis data; a process that converts analysis data to measurement data though the use of software metrics. Most mainstream analysis tools specify and examine these

software metrics through a relational database approach. In this case, an analyzable representation of the source code is generated, stored in a relational database, and queried through SQL requests to create measurement data (e.g., quality or maintainability metrics) [72].

However, for large software systems, the analysis data gets so enormous that existing approaches trade the amount of source code details stored for performance. This typically means an incomplete representation of source code where lower-level details are often omitted (e.g., method bodies, expressions, etc.). Although this strategy ends up facilitating scalability, it often limits the amount of supported measurements. In addition to being large, software systems also undergo continuous changes in order to adapt the new technology, to meet the new requirements, and to repair errors. These exponentially growing changes imply that software analysis tools have to continuously adjust their data models, restructure their relational database schemas, and reformulate complex SQL requests. Further, in order to formulate measurement data, end-users (e.g., quality experts, project managers, etc.) typically query the underlying database via queries on software metrics. This procedure requires not only an in-dept knowledge of the database schema but also expertise in the SQL querying language.

In order to address the above-mentioned concerns, we propose techniques and methods for the interactive visual analysis of software measurement data. To resolve scalability issues we advocate the use of a NoSQL approach that is based on a graph database. Using this methodology, we can perform complete analyses of large software systems and yet maintain a reasonable performance while creating measurement data. Furthermore, while approaching "big data" scalability can be maintained by distributing the underlying graph database across a multi-machine cluster. Additionally, to address changing requirements a graph database can be restructured far more easily due to an index-free adjacency and a schema-free data access [75]. In terms of the former, additional information such as the evolution of the software or the amount of effort needed to change software components may be added via links to new nodes. While in terms of the latter, the underlying graph model may be changed at run-time to store intermediate results without corrupting the analysis data model. Finally in terms of querying, whether it is SQL for relational databases or graph quering languages such as Gremlin or Cypher [179], we provide an interactive visual workflow modeling approach. Our approach makes it easier for non-experts to adjust existing metrics

or define new metrics without the prior know-how of the underlying database schema or the required database query language. Thereby alleviating end-users from this burden and instead empowering them with a means to visually specify their queries.

Irrespective of the database approach employed, the ensuing measurement data consists of metrics (numbers and sets of numbers) and is typically aggregated through statistical techniques, such as mean, variance, and standard deviation. However, numerical data by itself may not be so suitable to convey information in a comprehensible fashion. Instead, as an example it may be more appropriate to depict the distribution of a given metric across the software hierarchy in graphical terms. In this regards, we aim to provide the user with a plethora of options that include traditional methods as well as interactive software visualizations. Thus, providing them with a combined computational analysis and visualization approach.

Further, the workflow-based approach of our framework updates only the relevant measurement data and views when the user changes a metric – thereby, yielding an interactive exploration style of analyzing software measurements. In contrast, while using traditional approaches the complete measurement has to be evaluated again; a process that is too slow for large software systems with a complete set of analysis data.

It is for the reasons discussed here that we endorse a user-centric approach. Our proposed solution combines the specification and visualization of software measurements through data workflows. We aim to address the concerns of different stakeholders through the synergy of configurable data abstractions, metrics, and visualizations. Examples of such varied analyses include but are not limited to the performance, safety, security, or the quality of the system. Ultimately, the goal of our research is to empower end-users with the ability to apply tailored metrics and visualization metaphors to visually explore the characteristics of a software system according to their individual requirements.

In order to validate our ideas, we have developed a live-data prototype tool called VIMETRIK. Our preliminary study indicates the promise and feasibility of our approach to analyze software measurement data. In particular, our experiment shows that even graduate students with no knowledge of the underlying database model, querying mechanism, or software analysis could use our tool to generate and analyze some basic software metrics. The participants completed

these analysis tasks with a completion rate and accuracy of over 85%. We expect that if non-experts could use our tool then a professional software analyst would definitely benefit from the intuitive and easy-to-use means of understanding software measurement and analysis data.

3.2 Methodology

The main goal of a source code measuring tool is to provide a flexible analysis of a software system. However, current software analysis tools are limited by their lack of ability to process huge collections of multidimensional data sets. Many of these tools either do not perform a complete fact extraction of source code or only support a common set of software metrics. Others that provide a means of generating custom software metrics through queries tend to be quite difficult to use where the user requires extensive knowledge of the data ontology as well as the querying mechanism.

In this section, we present a methodology that aims to address these issues through a graph database model capable of capturing full details of a source code, a workflow-based approach to define metrics, and details of our live-data prototype VIMETRIK that combines these approaches in an intuitive and visual manner.

3.2.1 Metrics from a Graph Database

The main goal of a source code measuring tool is to provide a flexible analysis of a software system. As such, they let users pose queries about the system at different levels-of-detail in terms of software measurements or metrics; such as computing cyclomatic complexity, analyzing dependencies or call traces, detecting code smells, or using statistical analysis to find issues.

Often, due to memory limitations, these measurement tools use a relational database to store information about a system. Typically, these relational database approaches lead to a trade-off between scalability and the amount of source code details stored. On the one hand some systems store more details about the source code but do not scale to large programs, while on the other hand, others scale to large programs but provide limited information (either no method bodies or no details on expressions). Additionally these relational databases

are often faced with implementation related issues, such as: schema evolution over time and extensive joins of large tables. Furthermore, the restructuring of schema due to a change in the organization of the underlying data results in operations that are costly, complicated, and may not be performed automatically.

In contrast, we propose a graph database approach that aims to store full source code details, scales to large programs, and provides an easy means to restructure. In the recent past there have been similar attempts from researchers such as R.-G. Urma[2] and Ebert et al. [75], however, none have published a graph model that stores full source code information and is capable of addressing the many cross-cutting source code queries. In this regards, we present the benefits of employing a graph database over a relational database, provide details of our proposed graph model, and give a few examples of how software metrics may be gathered.

Benefits of a Graph Database

In the relational database paradigm, data is formally described and organized according to a database schema. Each database is a collection of related tables, where each table is a representation of an entity or object that is in a tabular format consisting of columns and rows. Columns are the attributes of an entity, while rows are the values or data instances. These databases implement an entity-relationship model where some data fields in these tables point to indexes in other tables; such pointers represent the relationships.

In contrast, the graph database paradigm uses graph structures with nodes, edges, and properties to represent and store data [180]. In this methodology, no index look-ups are required; every element (node) in the database contains a direct link (edge) to its adjacent element. Through such an index-free adjacency, graph databases can utilize graph theory for rapidly examining the connections and interconnections of nodes – an example of this is how Netflix recommends videos. As such, they scale more naturally to large data sets as they do not typically require the retrieval of connected records via the joining of common attributes. Studies such as the one conducted by Vicknair et al. [181] clearly indicate that graph databases are faster by a factor of up to ten times while traversing connected data.

[2] WiggleIndexer: Indexing of AST using graph databases (https://github.com/raoulDoc/WiggleIndexer/)

Graph databases also support semi-structured data as they do not depend on a rigid schema, this means they are more suited to manage ad-hoc and changing data with evolving schemas. This ability to quickly change the schema without requiring massive scripts can be a huge productivity boost for developers.

Interestingly, modeling data as a graph is natural and has the nice benefit of staying legible even for non-technical people. In the contect of software engineering the structure of object-oriented applications is a graph like structure. A good illustration of this is how different aspects of source code are modeled using ASTs, control-flow graphs, or parse-trees. For our work, we combine these aspects into a graph model or a Compound Attributed Graph (CAG) that serves as a syntactic and semantic model capturing the structure and interconnectivity amongst the different elements (e.g., nodes such as packages, compilation units, types, methods, fields, and expressions) of a software system.

Information from both relational and graph databases is retrieved via queries. While a relational database uses SQL to describe desired data, graph databases are accessed through query languages that express graph operations such as traversals and pattern matching. The former typically requires indicies (foreign keys) to perform extensive joins, while the latter provides an index-free access to adjacent elements.

Proposed Graph Data Model

In the software engineering domain, the abstraction principle is used to abstract parts of reality to reduce complexity and to allow efficient design and implementation of complex software systems. Similarly, compilers usually use ASTs to represent the structure of program code. They typically scan and parse program artifacts to extract such tree models. In general, trees are not designed to keep all necessary information about software engineering artifacts in an integrated form. Adding additional links between the vertices of a syntax tree, such as the ones found in control-flow graphs, leads to a more general graph like structure.

While our graph model is based on ASTs, it is tweaked to handle queries about software artifacts. The resulting Directed Acyclic Graph (DAG) contains additional links between the vertices of a syntax tree. It is our aim that by providing such a graph model, we may address the many cross-cutting queries that our domain experts may have.

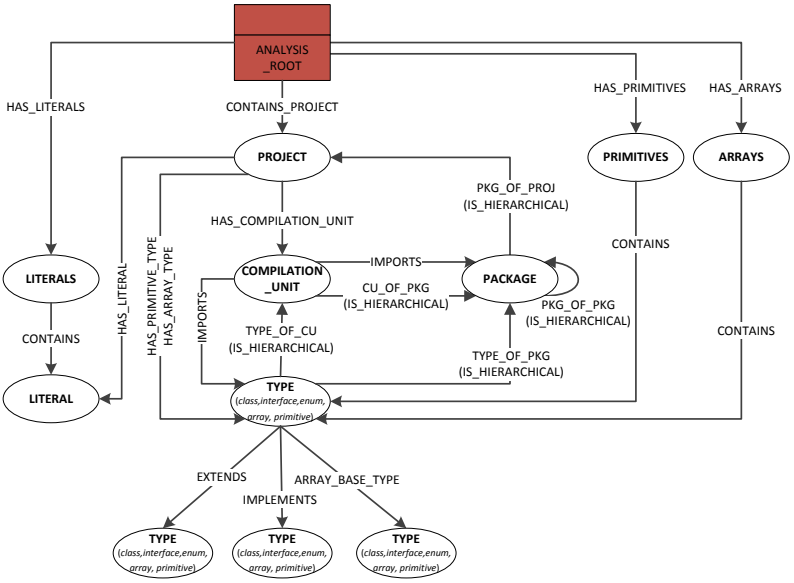
For the purpose of brevity, Figures 3.1, 3.2, and 3.3 show a few parts of this graph model, namely the top-level entities (project, package, compilation unit, etc.), some selected statements, expression, and member access as well as the links between these nodes.

In tracing some features of Figure 3.1a, we get an impression of not only the hierarchy of our top-level elements but also how additional information may be gathered. The key ingredients of the top-level model are: an analysis root, which may contain links to projects, primitives, arrays, and a literals root; projects, which may be connected to top-level packages or for convenience directly to compilation units; packages, which may have links to sub-packages; and compilation units, which contain a top-level type.

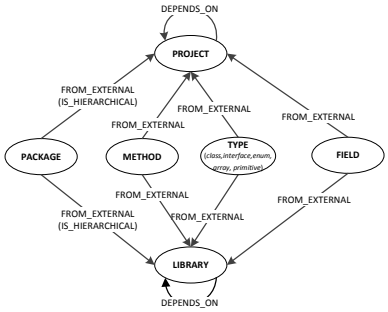
By traversing the above-mentioned nodes, we touch upon nodes and edges that form the top-level hierarchy of a software system. Additional edges between these elements provide further insight into the varied aspects of the software system. For example: IMPORTS, provides an understanding of the coupling between compilation units; HAS_LITERAL, offers a means to track the usage of literals in the project space; and EXTENDS and IMPLEMENTS, provide an access to base or parent classes and interfaces. Additionally, as Figure 3.1b shows some of these elements are handled differently as they may be found in external projects or libraries.

Similarly, by examining Figure 3.1c we get an impression of how the hierarchy of a compilation unit or source file is organized. Here, a BLOCK is a key element that is used to not only group statements but also as a body representation of classes, interfaces, enums, and methods. Each class, interface, or enum (TYPE) contains a body (BLOCK); each class body (BLOCK) may contain fields (FIELD), methods (METHOD), nested-types (TYPE), or static blocks (BLOCK); each type or method contains a body (BLOCK); and a method body may have top-level statements (STMT).

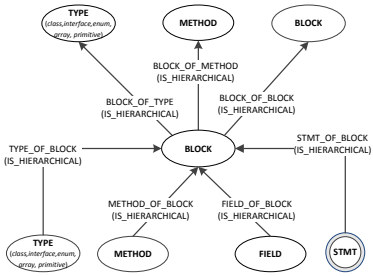
In the same way, top-level statements may be connected to other blocks, statements, or expressions found in the source code through node and edge representations. Here, it is important to mention that each type of statement has its own model; for example: an ASSERT_STMT has outgoing links to conditional and message EXPRESSIONs (see Figure 3.2a), a RETURN_STMT has an outgoing link to an EXPRESSION node (see Figure 3.2b), an EXPRESSION_STMT has outgoing links to an ASSIGNMENT, pre/post INC_DEC, or a MEMBER_ACCESS (see Figure 3.2c), and so on.



(a) Top-level graph model

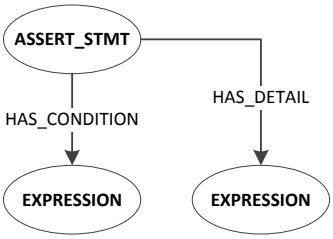


(b) External links

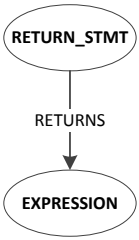


(c) Block graph model

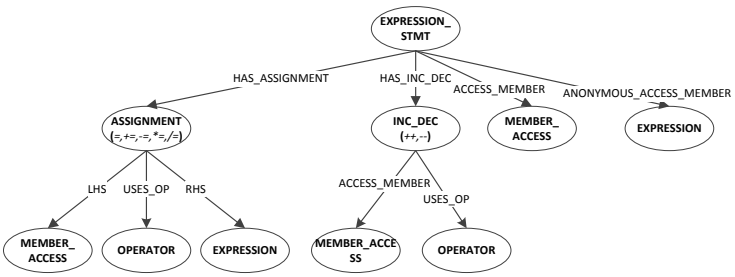
Figure 3.1. Graph model of top-level entitites



(a) Assert Statement



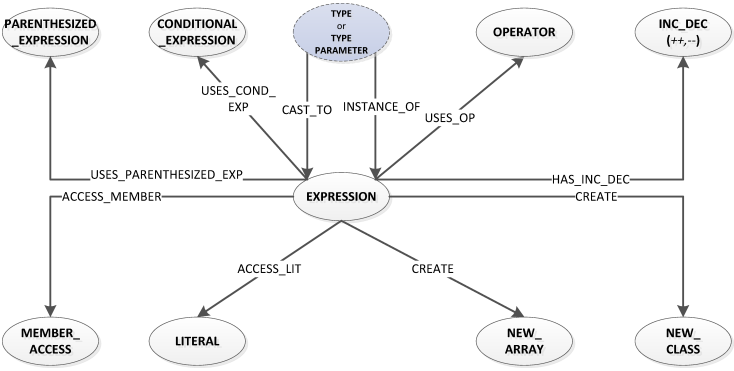
(b) Return Statement



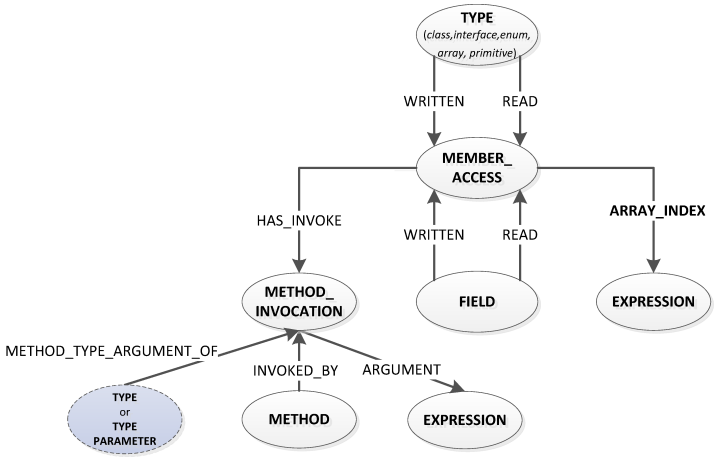
(c) Expression Statement

Figure 3.2. Graph model of selected statements

This forms a tree-like structure that not only contains the hierarchy of each source file, but more importantly links to elements found in other source files. These additional edges in our model connect related elements in a meaningful manner and are a key ingredient to cross-cutting queries, such as finding artifacts that connect method or variable usage to their declaration. Inspection of Figure 3.3 provides an insight into how such issues may be tackled. Here, expressions (EXPRESSION) are connected to a MEMBER_ACCESS that in turn provides an access path from a method or variable usage to its



(a) Expression graph model



(b) Member access graph model

Figure 3.3. Graph model of expression and member access

declaration. A variable usage is linked to its declaration via the READ and WRITTEN links, while a method usage is linked to its declaration via the path {MEMBER_ACCESS, METHOD_INVOCATION, METHOD}.

Furthermore, nodes and edges in our graph model have properties that allow us to distinguish the graph elements according to their types, their access parameters, or through other artifacts. In Table 3.1, we can see a complete listing of node and edge properties currently employed in our model. The most obvious of these properties is the type property which is depicted in Figures 3.1, 3.2, and 3.3 as node and edge labels. Each type of node may have additional properties, for example: a class or method may be private, public, or protected; a method may be overloaded; or an element may have comment or code lines associated with it. Similarly, edges have a type and may have isHierarchical and isExpressional properties, the latter two indicate whether an edge is a link of a syntax tree or one of our additional edges.

	Property
Node	name, type, handler, startLine, endLine, linesOfComment, isExternal, isClass, isEnum, isInterface, isPrimitive, isArray, isOverridenMethod, isMainMethod, isConstructor, isPrivate, isPublic, isProtected, isStatic, isTransient, isFinal, isAbstract, isNative, isThreadsafe, isSynchronized, isConstant, isVolatile, isStrictfp, isAnonymous
Edge	type, isHierarchical, isExpressional

Table 3.1. Properties within Graph Model

Finally, it is important to reiterate that as graph databases rely on semi-structured data, they can be easily modified in-order to cater to changing requirements. As a result, our model can be easily modified through the addition of new nodes, edges, or properties.

Extracting Source Code Facts

In order to make our tool readily available, we decided to develop on top of a well-known Integrated Development Environment (IDE). We choose to implement our prototype system as an Eclipse plugin as it is the most popular open source option and provides a wide

variety of features. One of these features, namely the Eclipse Java Development Tooling (JDT), enabled us to perform a full syntactical and semantic analysis of Java source code. For our prototype we describe a methodology that targets the Java programming language, however, our approach can easily be applied to other object-oriented programming languages that have an AST representation (e.g., C, C++, Smalltalk, etc.). It is important to note that eclipse already comes with a C/C++ Development Toolkit (CDT) that would allow us to perform a similar fact extraction of C or C++ programming languages.

The Eclipse JDT provides Application Program Interface (API)s to access and manipulate projects in the workspace. It allows access to Java source code via two different means, either through the Eclipse Java Model or the Eclipse AST. While the former is a light-weight and fault tolerant representation of source code, the latter is a more detailed tree representation where every Java source file is represented as a tree of AST nodes.

Information about the AST is gathered through the Visitor Pattern, a pattern that facilitates the capturing of information about source code elements through visitors. For example while parsing a source file a TypeDeclaration visitor method is called for every class declaration; a MethodDeclaration visitor method is called for every method declaration; and a VariableDeclarationFragment visitor method is called for every variable declaration. These methods provide access to specific information about the Java element they represent; such as the MethodDeclaration visitor that contains information about the name of the method, its return type, its parameters', etc.

Furthermore, various AST nodes have bindings that provide resolved information for several elements of the AST. These bindings not only provide a correlation between the AST and Java models, they also assist in determining to which declaration a reference belongs. For example, the MethodInvocation node returns a binding to the method that is invoked as well as a binding to the return type of the method. Additionally, these bindings also provide an access to a string representation of the elements' handle, which is stored as the handler parameter of our Neo4j nodes. We use this parameter to correlate between our Neo4j graph nodes and the JDT AST nodes while parsing the source code.

In our prototype source code querying system, the AST of each source code file is parsed thrice. In the first iteration, the AST visitors of each source code file translate AST elements and their properties to graph nodes in Neo4j according to our graph model. This forms an initial tree structure that contains a hierarchy of the source code. In the second and third iterations, we resolve bindings to transform this tree to a graph by adding additional links between related elements. Type hierarchy and attribution are handled in the second iteration, while data-flow is handled in the third.

Graph Model Query System

In order to highlight how source code metrics are realized, we present three examples of source code queries that are based on our graph model. These queries are encoded in the Cypher language of Neo4j, a declarative pattern matching graph query language that allows for expressive and efficient querying of the graph store. It supports adjacency queries, which test whether two nodes are connected to each other; reachability queries, which test whether a node is reachable from another node; pattern matching queries, which retrieve subgraphs containing the giving pattern; and summarization queries, which facilitate grouping and aggregation.

Typically, node patterns are represented in parentheses containing labels and/or properties, while relationship patterns are depicted using braces. The notation (n:Type { name: 'test.java'}) refers to a node that has a label Type and a property name that equates to test.java; in other words we get direct access to all source files called test. Similarly, the notation -[:TYPE]-> refers to a transitive closure of an edge labeled TYPE. A more detailed and up-to-date description of the Cypher query language is available on the Neo4j website[3].

In the first example, Query 3.1, we examine how to calculate the Depth of Inheritance Tree (DIT) metric that provides for each class a measure of the inheritance levels from the top-level object hierarchy. First, we generate a collection of all subgraphs starting at a type that extend or implement another type. Here we use transitive closure to specify a path that reaches a parent node of arbitrary length

[3] Neo4j - The World's Leading Graph Database (http://neo4j.org/)

as long as it exclusively consists of an edge labeled EXTENDS or IMPLEMENTS. Next, we return a reference to the given type as well as its depth in the inheritance tree. Finally, we sort the result in descending order and return the top ten matches.

```
MATCH (t:Type {type: 'TYPE'})
MATCH path = (t)-[:EXTENDS|IMPLEMENTS*]->(pt)
RETURN t, length(path) AS depth

ORDER BY depth DESC
LIMIT 10
```

Query 3.1. Depth of Inheritance Tree (DIT)

In the second example, Query 3.2, we look at how to extract the package hierarchy for a given software package. Similar to the previous query, we generate a collection of all package subgraphs rooted at the given package. However, we perform an optional pattern match as we want to include leaf packages whose result would be the empty set. Further, we use the extract collection function to return the handler parameter for each one of the child packages.

```
MATCH (pkg:Type {type: 'PACKAGE'})
OPTIONAL MATCH (pkg)<-[:PKG_OF_PKG*1..]-(child)

RETURN pkg, extract(c IN collect(child) :
                    c.handler) as childIds
```

Query 3.2. Extract package hierarchy

Finally, in the third example, Query 3.3, we compute the Efferent Coupling (Ce) for a particular method, i.e. we determine the number of methods that directly depend on a given method. Similar to the previous query, we first collect all subgraphs starting at a method and ending at the statements declared within. However, we add a WHERE clause to filter out hierarchical edges and to reduce the number of traversals. Next, a WITH clause is used to explicitly separate query parts and carry over the caller (ca) and statements (s) to the next part. In the next part, we switch our focus to the subgraphs that start at these statements and end at methods called through the INVOKED_BY relation. Finally, we return a reference to the caller and the number of callee methods.

```
MATCH (ca:Type {type: 'METHOD'})
MATCH (ca)<-[:BLOCK_OF_METHOD]-()<-[r*]-(s:Statement)
WHERE all(rel in r WHERE has(r.isHierarchical))

WITH ca, s

MATCH path = (s)-[r*]->(e:Expression)<-
             [:INVOKED_BY]-(ce)
WHERE all(rel in r WHERE has(r.isExpressional))

RETURN ca, length(path) AS Ce
```

Query 3.3. Compute the Efferent Coupling (Ce)

By adding additional filters to Query 3.3, we can get a handle on how many of these methods are declared in the same class or in a different compilation unit. Further, we can just as easily modify it to compute a "call graph" of a software system, i.e. determine the pair of methods which call each other.

The above examples show that Cypher is quite easy to understand and its syntax is easy for developers familiar with SQL. A comparison study conducted by Holzschuher et al. [179] shows Cypher performs quite well in comparison to other graph querying languages. However, as they explain in their study, there are also scenarios where graph quering languages struggle. In general, the performance is good for operations that can be formulated as one query (Queries 3.1 and 3.2). However, in other scenarios such as the one depicted in Query 3.3 this performances starts dropping drastically. In this particular case this happens due to the merging of two queries, somewhat similar to the joining of two tables. Other cases include Friend Of A Friend (FOAF) queries and for friend and group recommendations that require graph traversal operations over multiple levels.

Our experience shows that rephrasing queries using knowledge about the domain and application can improve performance. In particular, data should be filtered out as early as possible in order to reduce the amount of work that has to be done in later stages. Other performance enhancing improvements include using patterns that limit the portions of the dataset that needs to be explored and to return only data that is needed. However, due to the varied nature of potential queries, it is not always possible to restrict the amount of

data to be examined. Instead, for these types of queries, the native Neo4j Traversal framework[4] provides a much more robust and efficient means to specify desired movements through a graph and to capture required details.

While inquiring for software metrics via graph queries and traversals is handy, it requires an in-depth knowledge of the respective mechanisms (Cypher or Neo4j Traversal framework) and the underlying graph model. To alleviate this burden from non-expert users, we propose a workflow-based engine where the relevant queries or traversals are embedded in visual modules.

3.2.2 Using a Workflow-based Approach

The central focus of our work is to facilitate users that have no prior knowledge of the underlying graph model, database, or query mechanisms. As such, we investigated the use of a VPL as a means to break down larger software measurement concerns into smaller well-managed modules. The main emphasis of this approach is to build meaningful workflows, i.e., to concentrate more on the types of module, their configuration, and their interconnectivity rather than focusing on their inner workings (graph queries or traversals).

It is important to note that while we have implemented our ideas using the Konstanz Information Miner (KNIME) [108] open source environment, we could have also used a well-established commercial data pipelining tool such as InforSense KDE[5], Pipeline Pilot[6], etc. Instead, our focus is on providing a design that consists of modules which can be used to address software measurement concerns. A similar approach can also be implemented in tools much like the ones mentioned above.

In regards to switching to a workflow-paradigm from a textual querying system, we present details of the modules we provide, present some sample workflows that are used to gather software measurements, and highlight some useful tool-specific features.

[4] Neo4j Traversal Framework Java API (http://http://docs.neo4j.org/chunked/stable/tutorial-traversal-java-api.html)
[5] InforSense KDE (http://www.inforsense.com/)
[6] Pipeline Pilot (http://accelrys.com/products/pipeline-pilot/)

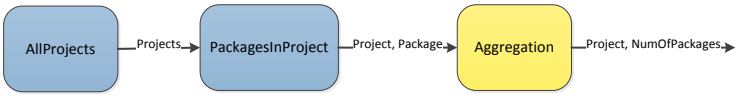


Figure 3.4. A simple workflow for number of packages

Software Fact-Extractors

The focal point of designing modules that process software entities, which we refer to as software *Fact-Extractors*, is to allow users to assemble and adapt an analysis flow compromised of standard building blocks. Thus, the main function of these modules is for users to combine them in a meaningful manner, through pipes that carry data, to produce useful software measurement results.

In Appendix 3.A.1, we present a table with a partial listing of the categories we have designed; i.e., Project, Package, Compilation Unit (CU), and Type. There are a number of Fact-Extractors under each category that typically expect a "category type" input, produce a specific result, and maybe connected to other modules. Additionally for each software module, the table provides a name, a description of its business logic, and its configuration options. For brevity purposes, we have not listed the remaining 25 modules under the Method, Field, Properties, and Viewer categories. Further, for extensibility purposes, we provide a generic query node where users may manually input Cypher query statements; thus, creating a custom software Fact-Extractor.

Designing Workflows

In order to highlight how the above-mentioned modules can be combined together in a workflow to produce a software measurement, we present some sample workflow designs. In these designs, blue nodes represent our software Fact-Extractors while yellow nodes symbolize data manipulators.

First let us examine the simple workflow of Figure 3.4 that calculates the number of packages per project. Since we are interested in projects that are a part of the source analysis and not in external libraries, we configure the AllProjects modules with the INTERNAL access option. From the user's perspective, this node produces a listing

of all the internal projects in the database and connects to the PackagesInProject module. However, internally it executes a Cypher query to produce these results. Similarly, the PackagesInProject traverses the PKG_OF_PROJ and PKG_OF_PKG links of Figure 3.1a to provide a project to package mapping. Finally, a data manipulation node labeled as Aggregation groups the projects according to a count of project to package mappings.

Figure 3.5 illustrates some more interesting workflow designs. In Figure 3.5a, we calculate the cyclic complexity of a method. In a similar manner to the previous example, the left half of the diagram accumulates some details regarding the internal methods of the database; it collects top level statements (TS) for each method, for each TS it gathers nested statements (NS), and then performs a join and merge operation on the results to produce a method to statement (Stmt) mapping. Next the user applies a filter to retain all statements that lead to a conditional loop; i.e., switch case, catch, do, for, if, while, etc. Additionally the user can also examine the expressions of the initial statements gathered for compound predicates such as && (conditional and), ||(conditional or), etc. Once all the statements and expressions of interest are collected, the results are aggregated to provide the cyclic complexity of each method – i.e., the number of independent paths through each method of the source code.

Similarly, in Figure 3.5b we calculate the response for a class. Initially, we collect all internal types present in the graph database and their methods. Next, we apply an InheritanceFilterForType to filter out inherited methods. Once we have the types we are interested in and their local methods, we gather all the invocations made to other methods and perform aggregations to count the Number of Local Methods (NumOfLMs) and the Number of Method Accesses (NumOfMAs). Finally, we add these numbers to produce the Response For a Class (RFC) coupling metric – i.e. the complexity of the class in terms of method calls.

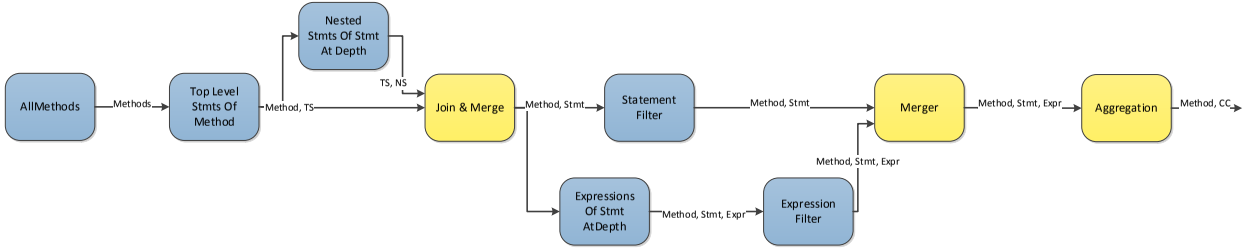
Useful tool-specific features

As stated earlier we built our prototype on top of KNIME, a visual data exploration and data mining tool that is written as a Java Eclipse plugin [108]. In this regards, we would like to point out some useful features that developers may benefit from.

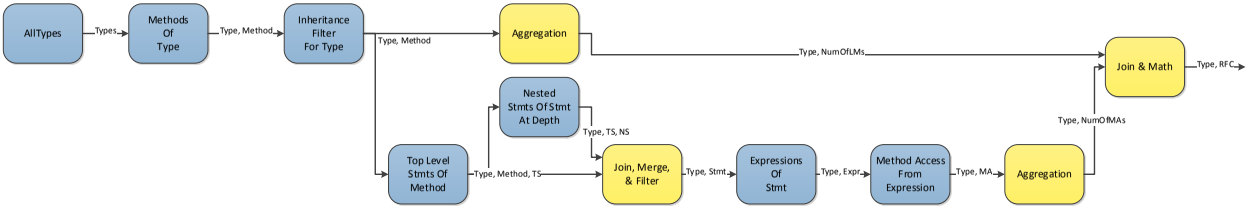
Typically, users arrange nodes or modules and connect them visually in the workflow editor with pipelines generating a dataflow network for analysis and visualization. What makes KNIME stand out amongst its competitors is a large community that provides nodes to incorporate existing tools, such as the R statistical analysis environment [104], the WEKA machine learning software [182], and the Cytoscape network analysis and visualization toolkit [183]. This is made possible by the extensible nature of KNIME where developers can write their own nodes to extend the built-in functionality.

The user models workflows in KNIME where nodes process data and transport that data via connections between nodes. These workflows usually start with a node that reads in data from some data source and stores them in a data structure called a DataTable; a structure quite similar to an SQL data table, which contains columns with a certain data type (integer, long, string, network, etc.). These data tables are then sent along connections to other nodes that modify, transform, model, or visualize the data. This includes a variety of data operations, such as filtering, merging, statistical functions (mean, standard deviation, etc.), and intensive data modeling operators.

KNIME models workflows as graphs connecting nodes as a DAG where the status of each node is tracked (i.e., error, configured, and executed). Nodes can be executed one at a time or an entire workflow can be executed such that the framework freely allocates the workload among parallel threads. A nice feature of this execution framework is that nodes first process the entire input table before forwarding the results to successor nodes. This provides a number of advantages, such as: each node stores its results permanently allowing the user to stop workflow execution at any node and resume it later on, immediate results can be scrutinized at any time, and new nodes can be introduced that readily access data produced by preceding nodes. In practice, while calculating software metrics for large systems this is a useful approach where modules not only depict intermediate results but also store them to disk. Another advantage of such an execution framework is that users may store and load complex workflows and their resulting data to and from disk. Thereby, avoiding time consuming re-execution for when the environment is restarted. Further, it provides a mechanism for users to share their measurement results.



(a) Cyclic Complexity (CC) of methods



(b) Response For Class (RFC)

Figure 3.5. More complex workflows

Other useful features worth mentioning are *meta-nodes*, *hiliting*, and numerous *data view's*. The former is a feature that we extensively employ to encapsulate discrete pieces of functionality into a single view. This provides us with two distinct advantages: 1) it enables us to design much larger complex workflows that are still readable by hiding complexity, and 2) the ability to reuse meta-nodes by copying them into different workflows. Hiliting allows users to select and highlight several rows in a data table and employ handlers that highlight the same data in other views. This would mean that measurement results that are above or below a certain threshold may be used to highlight the relevant software elements across multiple tables, visualizations, or both. Finally, data produced by each node in the workflow can be inspected at any point in time. The user simply has to connect one of the many data views to the output port and produce tabular views, scatter plots, box plots, histograms, parallel coordinates, bar charts, line charts, etc.

In spite of the various useful features found in KNIME there are still a few missing. Our experience shows that it would be highly beneficial to catalog user defined meta-nodes into a library; this would make it much easier for reuse rather than having to locate them in the created workflows. Another key feature missing is the automatic updating of successor nodes when the data of a predecessor node changes. An example of this might be when a particular node is reconfigured to produce different data, then using the current scheme in KNIME users have to either execute each successor node or "execute all" nodes in the current workflow. Instead, it would be useful to have a feature that automatically updates data in successive tables and views when a prior node or its data changes.

3.2.3 Live-data Prototype

In order to provide a clearer picture of the ideas presented in this paper, we report on a live-data prototype of an interactive visual analysis-tool called VIMETRIK. As Figure 3.6 shows, this prototype was built on top of the KNIME Eclipse plug-in.

As a prerequisite to working with our workflows the user is required to extract source code facts into a Neo4j graph database through a library we provide. This library extracts source code facts following the methodology described in Section 3.2.1. From the user's perspective, this is a simple process where the user loads Java projects into the

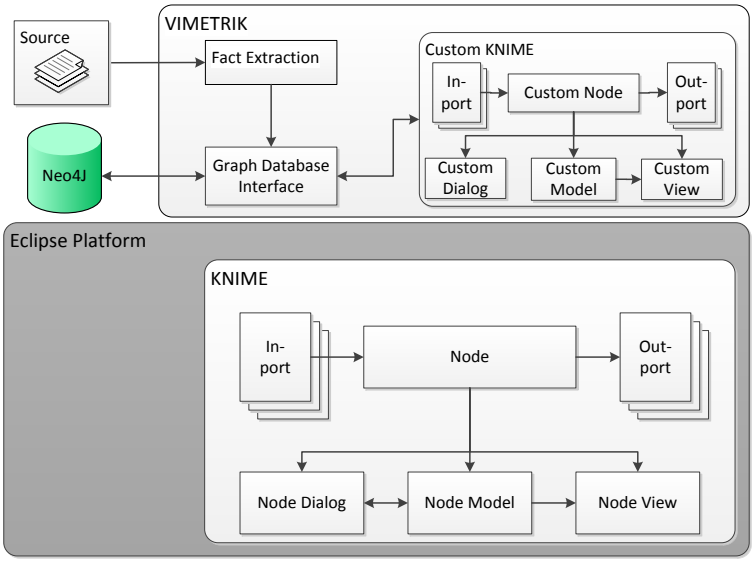


Figure 3.6. VIMETRIK prototype schema

Eclipse Java Package Explorer and clicks on "Extract Facts". In order to produce live-data for our prototype, we followed the same procedure to create a Neo4j graph database that contains source code facts for the well-known Apache Tomcat software system; resulting in a graph that consists of 235, 965 nodes and 375, 077 edges.

In KNIME the most common processing unit is a Node that represents each one of the visual workflow nodes. It follows the well-known Model-View-Controller design pattern by wrapping all the nodes' functionality into our custom implementations of a NodeModel, a NodeDialog, and one or more NodeView instances if appropriate. For our prototype we used the software Fact-Extractor designs that were described in Section 3.2.2 to implement nodes that either execute basic Neo4j Cypher queries or perform more complex graph traversals using the Neo4j Traversal framework. Our custom nodes gather the query and traversal results as DataTables and pass them to connected nodes to process or visualize. Additionally, these nodes provide a token that

contains details pertaining to the graph database connection. This approach enables us to process elements of our graph database and at the same time have the full functionality of all the nodes available in KNIME.

Interoperability between our software Fact-Extractors in KNIME and the Neo4j database is attained through the management of KNIMEs DataTable data structure and the specification of our graph queries. In terms of the former, we encode its meta-information with a custom DataTable specification that determines the column type; i.e., Project, Package, Compilation Unit, Type, Field, Method, Statement, or Expression. Each row of the DataTable contains a specific number of DataCell objects that holds the actual data in the form of long values or graph database node ids. The internals of each software module then perform graph traversals or pattern matchings based on these ids.

Further, for usability reasons and due to the nature of workflows, our software Fact-Extractors inspect the incoming table and automatically pick the last column of the required type; for example, MethodsOfType would automatically pick the last Type column available. Alternatively, the user may configure these nodes to pick another column of the appropriate type as input to the software module. If the type required is not present in the incoming data table, they produce an error message to alert the user. Additionally, we allow the user to interactively choose which columns from the input table get forwarded to the output table, append the results of our queries as columns, and provide a means to rename the output columns.

Using our prototype, we have developed several workflows that calculate metrics at different levels of the code structure; such as the Chidamber & Kemerer metrics suite [184], the quality extension of Chidamber & Kemerer metrics suite [185], Martin's metrics [186], the QMOOD metrics suite [187], McCabe's cyclic complexity measure [188], and standard size metrics such as Lines of Code (LOC), Lines of Comment (LC), Statement Count (SC), etc.

An example of such a workflow is depicted in Figure 3.7. The interconnectivity of our custom nodes (AllProjects, PackagesInProject, and CompilationUnitOfPackage) in the left half of the workflow is responsible for gathering the top-level hierarchy of the Apache Tomcat

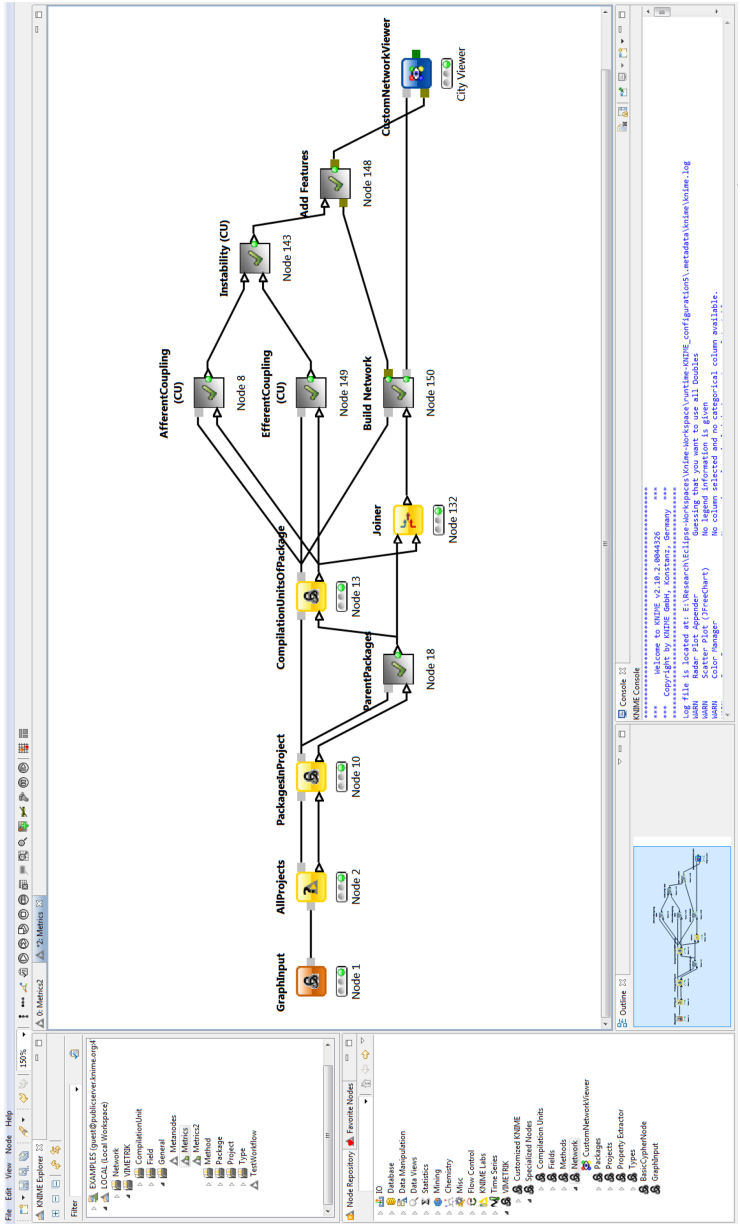
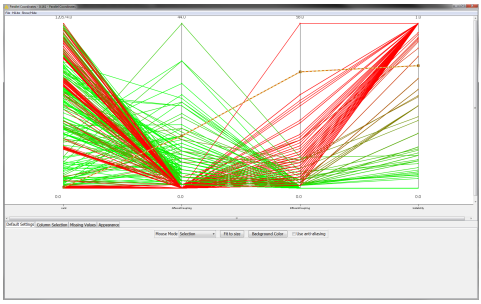
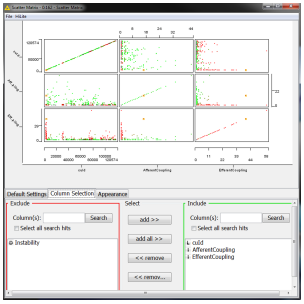


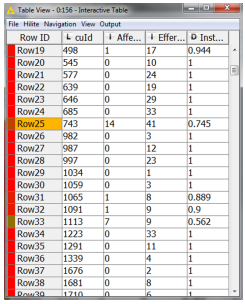
Figure 3.7. Sample VIMETRIK workflow



(a) Parallel Coordinates



(b) Scatter Matrix



Row ID	cuId	Affe...	Effer...	Inst...
Row19	498	1	17	0.944
Row20	545	0	10	1
Row21	577	0	24	1
Row22	639	0	19	1
Row23	646	0	29	1
Row24	685	0	33	1
Row25	743	14	41	0.745
Row26	982	0	3	1
Row27	987	0	12	1
Row28	997	0	23	1
Row29	1034	0	1	1
Row30	1059	0	3	1
Row31	1065	1	8	0.889
Row32	1091	1	9	0.9
Row33	1113	7	9	0.562
Row34	1223	0	33	1
Row35	1291	0	11	1
Row36	1339	0	4	1
Row37	1676	0	2	1
Row38	1681	0	8	1

(c) Tabular View

Figure 3.8. Standard views of measurement results using KNIME [108]

system. Once this information is attained we use meta-nodes (grey-boxes) to: 1) calculate the afferent (incoming dependencies) and efferent coupling (outgoing dependencies) of compilation units, and 2) feed this data into a network.

As described in Section 3.2.2, meta-nodes (grey boxes) are encapsulations of larger sub-workflows into a singular node representation. In the above example, EfferentCoupling gathers all the types imported by each compilation unit (ImportsOfCompilationUnit), extracts the parent compilation unit of these types (DeclaringCompilationUnit), and then filters and counts the outgoing dependencies that are part of the Apache Tomcat source code.

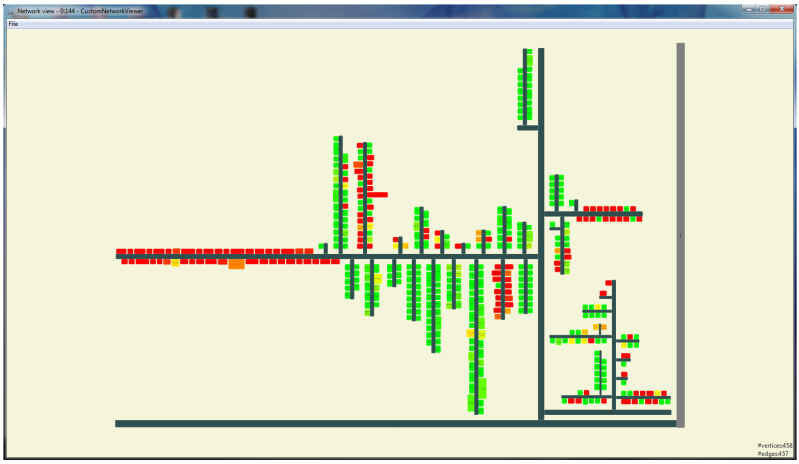


Figure 3.9. Custom view of the measurement results

We then calculate the instability, a value in the range between 0.0 and 1.0, as the ratio of efferent to total coupling to get an indication of the compilation unit's resilience to change. The afferent coupling, efferent coupling, and instability metrics are then added as features into our network and provided as input to our custom network viewer.

The next step in the analysis is the sense-making process. In traditional approaches, this means examining the measurement data across multiple views. In Figure 3.8, the above-mentioned metrics are presented using standard KNIME views; Figure 3.8a uses parallel coordinates, Figure 3.8b uses a scatter matrix, and Figure 3.8c uses a tabular view. The analyst follows the CMV approach of "brushing and linking" elements to get examine the afferent coupling, efferent coupling, and instability metrics. However, a key ingredient missing is the context; i.e. to answer questions such as which package contains the majority of unstable compilation units.

Alternatively, we provide interactive software visualizations that are configured via the CustomNetworkViewer node. In our current implementation, we allow the user to configure the type of visualization (city, sunburst, or hyperbolic), apply some generic settings, and map features as entity properties. Figure 3.9 shows one of these possibilities, where we have encoded the measurements mentioned above in a city view [57]. Here, the package hierarchy is represented through streets with alternating vertical and horizontal orientations and source files

or compilation units are symbolized by plots. Further, the viewer has been configured to map the efferent coupling values as the width, the afferent coupling values as the height, and the instability values as the color of our plots. The latter employs a traffic-light metaphor that interpolates values in the [0.0, 0.5, 1.0] range to the colors [green, yellow, red]; the green color represents maximally stable (0.0), yellow depicts semi-stable (0.5), and red signifies maximally unstable (1.0) source files. Interacting with the city view tells us that 118 out of 411 source files would have a significant impact on other source files if they were to be modified; and that the majority of these unstable source files are located in the *org.apache.catalina* package.

3.3 Experiment

We propose VIMETRIK for improving the understanding of large complex software systems. In particular, we assume that through the use of our workflow-based approach of specifying and visualizing user-defined software measurements, software analysts would be able to perform measurement tasks effectively, efficiently, and importantly without dealing with the complexity of the data ontology or querying mechanisms. We also expect VIMETRIK to be well accepted and perceive it to be a useful tool for the analysis of large complex software systems. As a proof of concept and to test our assumptions, we designed and performed a preliminary study.

3.3.1 Research Purpose and Hypothesis

In order to evaluate our design we performed a preliminary study on groups of experts in the domains of Software Engineering (SE), Computer Graphics (CG), and Databases and Information Systems (DBIS). These experts, graduate students, were chosen from related fields to examine if the varied expertise had any effect on the experiment results. Ultimately, the primary goal of our preliminary study was to both validate the individual elements of our design as well as the hypotheses that using the VIMETRIK approach users with varied levels of expertise would perform above a certain effectiveness (completion and accuracy) threshold. Thus, we defined the following research hypotheses:

- H_1: The VIMETRIK framework can facilitate users with no prior experience to achieve a completion rate of 85% or more for software analysis tasks.
- H_2: The VIMETRIK framework can facilitate users with no prior experience to achieve an accuracy of 85% or more for software analysis tasks.

The secondary goal was to compare the results of each user group with respect to effectiveness, efficiency, and user satisfaction. Therefore, we define four additional research hypotheses:

- H_3: On average, all three groups produce equal completion rates for the same software analysis tasks.
- H_4: On average, all three groups achieve equal accuracy in completing the same software analysis tasks.
- H_5: On average, all three groups produce equal efficiency in completing the same software analysis tasks.
- H_6: Users agree that the VIMETRIK framework is indeed acceptable and useful for the analysis of large complex software systems.

3.3.2 Operationalization

In order to test the above hypotheses, we operationalized four variables of interest, selected a software system to be analyzed, and designed the tasks to be performed. First, the variables of interest are operationalized as follows:

i. Completion is measured as the percentage of completed tasks. If a task requires three Fact-Extractors or nodes to be configured and the user only manages one then the completion rate is one-third or 33.33%. The same completion rate holds if two erroneous nodes are used. Further, 5% is deducted for each faulty node configuration.
ii. Accuracy is the closeness of measurements of a task to that task's true value. If a task consists of three measurements and the user records the wrong value for one of the measurements, an accuracy of 66.67% is awarded.
iii. Efficiency is the time required for accomplishing a set of tasks.
iv. User satisfaction is a combination of acceptability and usability of the VIMETRIK framework (see Appendix 3.A.2). Acceptability is measured using the Technology Acceptance Model (TAM) model [189], where 4 questions focused on performance and effort

expectancy are selected. On the other hand, usability is measured based on the work of Nestler et al. [190], where 11 questions based on utility, intuitiveness, learnability, and personal effect are selected. Each question is rated using a five-point Likert scale (1: I strongly disagree, 5: I strongly agree).

Second, we selected the Apache Tomcat system as the system to investigate. Finally, software understanding (measurement) tasks were classified into three categories, i) *lightweight tasks* (configuring and connecting one Fact-Extractor node to the workflow), ii) *intermediate tasks* (configuring and connecting two Fact-Extractor nodes to the workflow), and iii) *advanced tasks* (configuring and connecting three or more Fact-Extractor nodes to the workflow). Additionally, each one of these tasks required performing data operations such as aggregating, grouping, and merging results. In Appendix 3.A.3, we present a complete listing of all these tasks.

3.3.3 Field Test

A field test was conducted with two experts; one of the experts was from the field of software engineering and the other from the field of human computer interaction. The purpose of this test was to get an early feedback regarding the experimental design, the selected tasks, and the time required for solving them.

The software analyst completed the measurement tasks in an hour. He considered the tasks to be realistic and found VIMETRIK to be a very useful tool for the analysis and understanding of software systems. Working with workflows instead of textual query statements was found to be easier, less error prone, and more intuitive. Further, he felt he was more in control of the analysis and could easily backtrack errors by inspecting the results of intermediate nodes. He really liked the idea of meta-nodes and felt they not also furthered reusability but also helped in keeping the workspace "tidy". However, he recommended leaving them out of the experiment and focus more on normal workflows to generate software measurements. The other recommendation he made was to reduce the number of tasks to ensure that a "non-expert" is not overwhelmed.

The second expert solved the original measurement tasks in one hour and twenty minutes. Although the second expert was not a software engineer, he felt the tasks were representative of those typically performed by a software analyst. He found VIMETRIK to be a pow-

erful visual tool that was well structured and allowed him to perform basic software analysis tasks even without the necessary know-how. However, he felt there was a learning-curve involved where he had to get used to the different node types and the configuration options. In this regards, he recommended not to describe all the features at once but rather separately for each type of task. Secondly, he felt that one of the lightweight tasks should be removed from the experiment and instead be used for training purposes. Additionally, like the first expert he felt the experiment was a bit too long and that it should be kept to an hour at maximum.

Consequently, the original set of tasks were tweaked to address the experts' feedback. On the one hand, one of the lightweight tasks (Task 1.1) was removed from the experiment and was used to train the participant instead. While on the other hand, we removed some measurement tasks (afferent and efferent coupling of compilation units) from the experiment. The original design was also restructured into fact extraction (Task 1 and 2), measurement (Task 3), and visualization (Task 4) tasks, where each task consisted of two to three subtasks. The participant was then introduced to the required features prior to each task instead of all at once.

3.3.4 Preliminary Study

Subjects

The preliminary study was conducted with 21 participants, all of whom where graduate students at the University of Kaiserslautern. This experiment followed a quasi-experimental design where the 21 participants were evenly divided into three groups of 7 participants each (i.e. 7 from the SE faculty, 7 from the CG faculty, and 7 from the DBIS faculty). None of the participants had any prior experience with VIMETRIK, KNIME, or visual queries. However, 19 students had varying degrees of expertise in SQL. Four of these students were females and all of them were required to have object-oriented programming, object-oriented design, and basic query knowledge.

Experimental Setup

The same software system, Apache Tomcat, was analyzed by all the participants. All of them performed the same software measurement tasks using the same working environment. This working environment consisted of a dual-monitor desktop with an Eclipse based KNIME Software Development Kit (SDK) enhanced with our VIMETRIK nodes (Fact-Extractors, CustomNetworkViewer, and some utility nodes). The resulting VIMETRIK workspace was prepared with the same data; a Neo4j database containing facts of the Apache Tomcat system according to our graph model, 54 VIMETRIK nodes, 9 "Data View" nodes, and KNIME data manipulator nodes (i.e., Joiner, GroupBy, etc.).

Each workspace was completely restored at the beginning of each experiment session. The participants were given new workflow sheets with a pre-configured GraphInput node and a BuildNetwork meta-node. The former was configured with the location of the graph database, while the latter was used in the visualization task to convert software measurements into a network. Each participant was provided with an introduction handout and subsequently given a small tutorial to collect all the packages within the Apache Tomcat system. In this tutorial, a walkthrough was provided to: 1) first access all the Apache Tomcat projects via the Graph Input node, 2) second access all the packages via the projects founds, and 3) group the results to get a count of the total number of packages. At the end of the walkthrough, the participants were asked to replicate the above-mentioned workflow before starting with the experiment (see Appendix 3.A.3). Additionally, in between each task participants were afforded a similar walkthrough that introduced them to new features required for the next task.

Each participant was allowed unlimited time to finish the tasks and to fill the questionnaire regarding User Satisfaction. The resulting materials were then collected and analyzed.

Data Collection

The time and the answer to each question (see Appendix 3.A.3) was collected using an exercise sheet. Additionally, a printed questionnaire (see Appendix 3.A.2) was used to eliciting the perception of the participants regarding the user satisfaction of the VIMETRIK tool. The participants were also solicited for some open-ended questions to improve our framework. These questions consisted of the following:

- What are the positive things you found in the visual query approach as compared to standard text querying?
- What are the drawbacks in the current system?
- What are your recommendations to improve the current system?
- Do you have any other comments?

Data Analysis

A transcript of the collected completion, correctness, efficiency, and user satisfaction data was compiled in excel. The data of each subject is kept anonymous and confidential. Further, the completion and correctness task results were weighed according to their difficulty: lightweight tasks were assigned a weight of 1, intermediate tasks were accredited a weight of 3, and advanced tasks were given a weightage of 5.

We applied descriptive statistic methods such as the sample mean, standard deviation, median, and range to the experimental data. Additionally, we used the MegaStat excel plug-in to statistically test our hypotheses using the Wilicoxon T-Test and one-way ANOVA-Test. While the Wilicoxon T-Test is a non-parametric statistical test that can be applied to non-normalized test results for hypotheses H_1, H_2, and H_6, the ANOVA-Test is a better fit for hypotheses H_3, H_4, and H_5 to compare results across more than two groups. We performed the tests with a confidence level for rejecting the null hypotheses at 95%.

3.4 Results

The statistical evaluation of the preliminary study is presented in this section. In this regards, we shall examine the effectiveness (completion and correctness), efficiency (time required for tasks), and the user satisfaction experiment results. Further, we argue for the earlier stated hypotheses by performing hypothesis testing (Wilicoxon T-Test and one-way ANOVA) on the collected data. Finally, we conclude this section by examining the threats to validity.

3.4.1 Effectiveness Results

The completion rate and accuracy with which participants completed software measurement tasks was measured to examine the effectiveness of the VIMETRIK framework. In particular, we wanted to study the effectiveness of participants with a varied set of expertise that had no prior knowledge of our tool. The participants were expected to complete more than 85% of the tasks without any assistance and achieve an accuracy of 85% or more while performing basic software measurement tasks.

Completion Results

The participants were asked to complete 9 subtasks with varied level of difficulties. For each task they were expected to configure and connect one or more nodes to the workflow. There was no assistance provided unless the participant could not proceed further on his own, in which case hints were provided and completion points were deducted. Further, they were penalized 5% for each faulty node configuration. In order to minimize the effect of cascading errors, nodes and their configurations were adjusted after each task. The participants were awarded 0% to 100% for the completion of each subtask, each subtask was weighed according to its difficulty (lightweight = 1, intermediate = 3, and advanced = 5), and a value between 0% and 100% was computed for the completion of all tasks. The results are presented in the box-plots of Figure 3.10. Based on these results, the following observations can be made:

- The average completion rate for SE and DBIS students differs more than the score of the CG students.
- On average, the completion rate for CG students was slightly higher.
- Independent of the user group, participants on average completed 90% or more of the tasks without any assistance.

The mean, median, and standard deviation values of the completion rates with which the participants performed are presented in Table 3.2. The results have been broken down according to task difficulty to inspect the completion rate for each difficulty level. It is not surprising to see comparable completion rates across the three groups as none of the participants had prior exposure to the VIMETRIK framework.

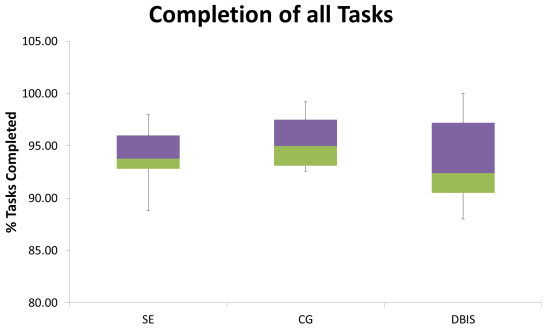


Figure 3.10. Box-plots of Completion rates

Further, it was also expected that they can complete lightweight and intermediate tasks without assistance, while for advanced tasks these completion rates drop slightly. In general, participants felt they needed to get used to the tool in order to join and merge results when adding more than two nodes to the workflow.

		Mean	Median	Std. Deviation
lightweight	SE	98.21	100.00	3.66
	CG	95.54	100.00	7.74
	DBIS	99.46	100.00	2.08
intermediate	SE	95.36	100.00	8.43
	CG	98.21	100.00	5.41
	DBIS	96.07	100.00	5.94
advanced	SE	92.38	100.00	7.18
	CG	94.29	100.00	7.46
	DBIS	91.19	90.00	7.23

Table 3.2. Participants Completion rate (% of tasks completed)

The corresponding completion hypotheses were tested using the Wilicoxon T and one-way ANOVA tests. These results indicate that on average participants completed 85% of the tasks and that there was no significant effect on the completion rates for the three user groups.

$H_{0,1}$: $\mu = 85\%$
$H_{a,1}$: $\mu > 85\%$
Test input: the completion rate of the task results for all participants
Test: Wilicoxon T Test
Result: Z=4.04, p=1.0E-04, $p < 0.05$; thus, the null hypothesis is rejected

$H_{0,3}$: $\mu_{SE} = \mu_{CG} = \mu_{DBIS}$
$H_{a,3}$: not all μ's are equal
Test input: the completion rate of the task results for each group
Test: ANOVA (one-way)
Result: F(2, 18)=0.46, p=0.6409, $p > 0.05$; thus, the null hypothesis is accepted

Accuracy Results

The accuracy with which each participant completed a subtask was measured as the closeness of measurements of a subtask to that subtask's true value. The participants were awarded 0% to 100% for the accuracy of each subtask, each subtask was weighed according to its difficulty (lightweight = 1, intermediate = 3, and advanced = 5), and a value between 0% and 100% was computed for the accuracy of all tasks. The results are presented in the box-plots of Figure 3.11. Based on these results, the following observations can be made:

- The average accuracy rate for DBIS students differs more than the score of the SE and CG students.
- On average, the accuracy rate for CG students was slightly higher.
- Independent of the user group, participants on average scored an accuracy rate of 85% or more.

The mean, median, and standard deviation values of the accuracy rates with which the participants performed are presented in Table 3.3. Again, to examine the level of accuracy reached for each difficulty level, the results have been broken down according to task difficulty. Our findings show that for lightweight tasks CG and DBIS students performed better than SE. This is attributed to the SEs needing more time to get used to visual queries as compared to textual queries. Further, SE and CG students performed better than DBIS students

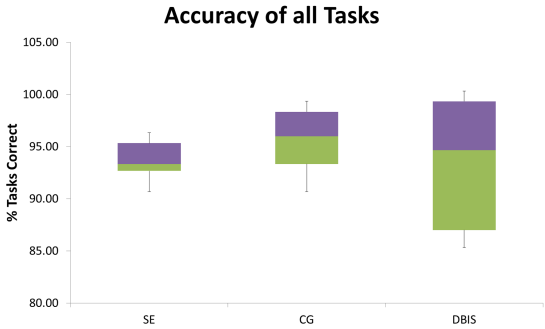


Figure 3.11. Box-plots of Accuracy rates

for intermediate tasks. We believe this to be an artifact of joining and merging results using additional nodes as compared to a single SQL query that combines the two. Further, it was also expected that the participants improve as they get accustomed to the tool. This can be seen by the similarly high accuracy rates achieved for more advanced tasks.

		Mean	Median	Std. Deviation
lightweight	SE	77.38	100.00	39.60
	CG	87.50	100.00	27.36
	DBIS	90.48	100.00	23.76
intermediate	SE	92.86	100.00	19.30
	CG	90.48	100.00	24.21
	DBIS	82.14	100.00	37.25
advanced	SE	98.41	100.00	7.27
	CG	100.00	100.00	0.00
	DBIS	98.14	100.00	7.27

Table 3.3. Participants Accuracy rate (% of correct task results)

The corresponding accuracy hypotheses were tested using the Wilicoxon T and one-way ANOVA tests. These results indicate that on average participants achieved an accuracy of atleast 85% and that there was no significant effect on the accuracy rates for the three user groups.

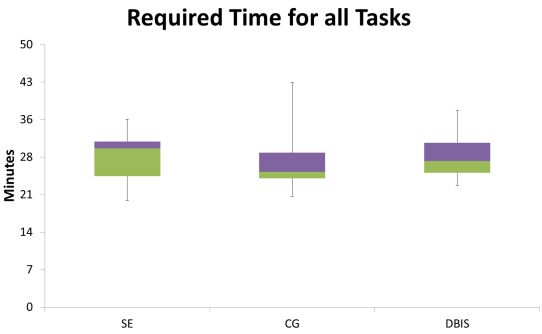


Figure 3.12. Box-plots of Efficiency results

$\mathbf{H_{0,2}}$: $\mu = 85\%$
$\mathbf{H_{a,2}}$: $\mu > 85\%$
Test input: the accuracy rate of the task results for all participants
Test: Wilicoxon T Test
Result: Z=4.12, p=3.78E-5, $p < 0.05$; thus, the null hypothesis is rejected

$\mathbf{H_{0,4}}$: $\mu_{SE} = \mu_{CG} = \mu_{DBIS}$
$\mathbf{H_{a,4}}$: not all μ's are equal
Test input: the accuracy rate of the task results for each group
Test: ANOVA (one-way)
Result: F(2, 18)=0.60, p=0.5589, $p > 0.05$; thus, the null hypothesis is accepted

3.4.2 Efficiency Results

The time required for each participant to complete a subtask was measured to compare the efficiency with which the participants performed in each group. The time for each task was an accumulation of the time required for the respective subtasks. Similarly, total time is an accumulation of the time required for all tasks. The results are presented in the box-plots of Figure 3.12. Based on these results, the following observations can be made:

- There is a larger variance from the average required time for CG students as compared to the SE and DBIS students.

- Members of the SE, CG, and DBIS groups were equally efficient.
- Independent of the user group, participants on average required around 30 minutes to complete the tasks.

The mean, median, and standard deviation values of the efficiency with which the participants performed are presented in Table 3.4. Again, to examine the level of efficiency attained for each difficulty level, the results have been broken down according to task difficulty. Although there is a statistical difference between the time required for each type of task (lightweight tasks $\sim$ 2 minutes, intermediate tasks $\sim$ 2.5 minutes, and advanced tasks $\sim$ 5 minutes), there is not much difference in the required times across the three user groups. This observation can be explained by the fact that none of the participants had prior exposure to our tool. Further, this observation also supports hypothesis H_5.

		Mean	Median	Std. Deviation
lightweight	SE	1:47	1:31	0:51
	CG	1:49	1:38	0:27
	DBIS	1:57	1:49	0:48
intermediate	SE	2:42	2:34	0:57
	CG	2:33	2:22	0:43
	DBIS	2:50	2:31	1:09
advanced	SE	5:24	4:37	1:30
	CG	5:21	4:40	2:45
	DBIS	5:13	4:57	2:06

Table 3.4. Participants Efficiency (time required in mm:ss)

In order to statistically test the above-mentioned hypothesis, we applied the one-way ANOVA test to the efficiency data. The results of this test confirm that the user's expertise had no bearing on the time required to complete our evaluation tasks.

$H_{0,5}$: $\mu_{SE} = \mu_{CG} = \mu_{DBIS}$
$H_{a,5}$: not all μ's are equal
Test input: the time required to complete tasks
Test: ANOVA (one-way)
Result: $F(2, 18)=0.16$, $p=0.8532$, $p > 0.05$; thus, the null hypothesis is accepted

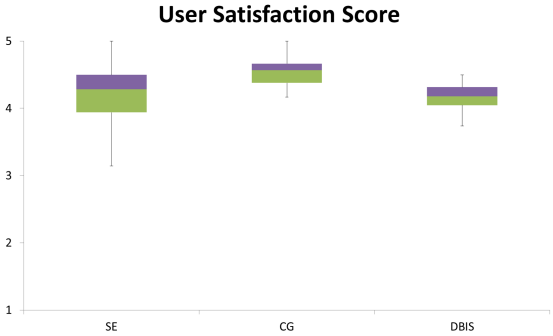


Figure 3.13. Box-plots of User-Satisfaction results

3.4.3 User Satisfaction Results

The participants of the experiment were asked to give their personal assessment on how satisfied (both acceptability and usability measures) they felt with the VIMETRIK tool. As mentioned earlier, we asked 4 questions from the performance and effort expectancy parameters of the TAM model and 11 questions from the utility, intuitiveness, learnability, and personal effect parameters from the work of Nestler et al. [190]. The participants responses ranged from 1 (Strongly Disagree) to 5 (Strongly Agree) for each question. By examining Figure 3.13, we can make the following remarks about the user satisfaction scores for the different user groups:

- The average user satisfaction score of the SE students differs more than the scores of the CG and DBIS students.
- On average, the user satisfaction score of the CG students was higher than the SE and DBIS students.
- Independent of the user group, participants on average gave a score of around 4 out of 5 for user satisfaction.

The aggregated mean, median, and standard deviation values for the parameters mentioned and the average overall user satisfaction score are presented in Table 3.5. These results indicate similar responses for 5 out of the 6 parameters of user satisfaction. In contrast, there was a more varied response for the Learnability parameter. The

participants felt that although the VIMETRIK tool was quite powerful tool, it was a bit complicated in terms of joining and merging query results. However, they also felt that this complexity could be overcome through further exposure and repetitive usage of the tool.

		Mean	Median	Std. Deviation
Performance E.	SE	4.14	4.00	0.63
	CG	4.71	5.00	0.39
	DBIS	4.29	4.50	0.49
Effort E.	SE	4.29	4.50	0.91
	CG	4.43	4.50	0.35
	DBIS	4.36	4.50	0.48
Utility	SE	4.43	4.50	0.45
	CG	4.93	5.00	0.19
	DBIS	4.43	4.50	0.35
Intuitiveness	SE	3.97	4.00	0.59
	CG	4.41	4.40	0.42
	DBIS	3.85	3.80	0.46
Leanability	SE	3.86	4.00	1.07
	CG	4.29	4.00	0.76
	DBIS	4.00	4.00	0.82
Personal Effect	SE	4.43	4.33	0.50
	CG	4.50	4.67	0.50
	DBIS	4.05	4.33	0.71
Overall	SE	4.19	4.28	0.61
	CG	4.55	4.57	0.29
	DBIS	4.16	4.18	0.26

Table 3.5. Participants User Satisfaction response

The corresponding user satisfaction hypothesis was tested using the Wilicoxon T test. These results indicate that the participants were on average highly satisfied with our approach to generate and visualize software measurements.

$H_{0,6}$: $\mu = 4$
$H_{a,6}$: $\mu > 4$
Test input: results of the user satisfaction questionnaire
Test: Wilicoxon T Test
Result: Z=2.70, p=6.9E-3, $p < 0.05$; thus, the null hypothesis is rejected

3.4.4 Threats to Validity

Threats to internal and external validity are discussed in this section. Threats to internal validity refer to conditions that constrain the confidence level of the results, while threats to external validity are conditions that limit the generalization of the study.

Internal Validity

There exist a few internal validity threats that relate to the participants used in this experiment and the selection of software measurement tasks.

With respect to the participants, it could be questioned if they were sufficiently competent. We have reduced this threat through a priori assessment of the participants' competence in the relevant fields, which pointed out that all subjects had at least an elementary knowledge of object-oriented programming, object-oriented design, and basic queries. Further, none of the participants had prior knowledge of VIMETRIK, KNIME, or visual queries. All the participants underwent the same walkthrough and training procedure. The experiment data was also compared to the software analyst's field test result to make sure the experiment results were representative. Our statistical results show that there was not a lot of difference in the dependent variables within each group.

It could also be argued that the participants were not properly motivated or may have had too much knowledge of the experimental goal. The former threat is mitigated by the fact they all participated on a voluntary basis; as for the latter, the participants were not familiar with the actual research questions or hypotheses.

In terms of the tasks, they were designed after consulting with an expert software analyst. Asking another expert or a group of experts would quite possibly produce another set of tasks. It can also be said that only a limited set of size and coupling measurements were considered that could adversely effect the results. The rationale behind our decisions is that we were looking for some representative tasks that could be completed in a reasonable amount of time. The purpose of our preliminary study was to conduct a preliminary analysis to gauge the feasibility of our workflow-based approach; i.e., to get a quantitative proof that the system has potential to succeed on a full-scale basis.

Similarly, we argue that the Apache Tomcat servlet container is representative of a real-world software system. In the future, we plan to incorporate more experts, analyze a more complex software system, and conduct a comparative study of related software analysis tools.

External Validity

The generalizability of our results could be hampered by the limited representations of the participants, the tasks, and Apache Tomcat as the only subject system.

In terms of the participants, the use of professional software analysts instead of Ph.D. candidates and M.Sc. students could have yielded different results. Unfortunately, it is quite difficult to motivate people from industry to sacrifice an hour of their precious time. Nevertheless, compared to studies that often employ undergraduate students, we assume the expertise levels of our 21 subjects to be relatively high. Furthermore, an overarching goal of the pilot studies was to test if the workflow-based interface was easy and intuitive enough for even non-experts to use. It is expected that if a layman could use VIMETRIK to generate software measurements, then a professional software analyst would definitely be able to do better (we intend to confirm this in a later research study).

Another external validity threat concerns the software metric tasks (Task 3), which may not reflect real software understanding situations. We tried to neutralize this threat by relying on our expert's feedback, which is based on the activities found in software analysis and the well-known efferent coupling and cyclic complexity measures.

Finally, the analysis of a different system (or additional runs) may have yielded different results. Apache Tomcat was chosen as it is a well-known real-world system that the participants have heard of; finding another system that the participants could relate to is not trivial. An additional system to analyze or extra runs would impose added burden on the participants or require more of them. While this may have been feasible in case the groups exclusively consisted of students, it is not realistic in the case of Ph.D. candidates as they have little time to spare.

3.5 Concluding Remarks

A deep understanding of a software system's design space is a significant factor for its success. In this regards, software measurements can impart invaluable insights into the nature of a software system and provide a means to make informed decisions regarding its design or implementation in terms of quality, reliability, dependability, or performance. However, this process is convoluted in practice due to the lack of measurement methodologies and the difficulty and debate around which metric to use. This makes it difficult to produce software analysis tools that address value drivers and way of working of the various user groups.

To address these issues we propose a user-centric approach that 1) allows users with no-prior knowledge of our underlying graph model or query mechanisms to develop custom measurements through interactive data workflows, and 2) the ability to present measurement-related information in a comprehensive manner. Both scenarios provide users with the complete freedom to define and view software measurements as per their requirements. We believe our approach as a means to effectively produce and inspect measurements of value for different stakeholders.

The preliminary study we conducted shows both the feasibility and promise of our proposed solution. In particular, participants with no knowledge of the underlying database model, querying mechanism, or software analysis could use our tool to understand software measurement and analysis data. The intuitive and easy-to-use nature of our approach is highlighted by the fact that independent of the participants' expertise, they were able to complete basic software analysis tasks with a completion rate and accuracy of over 85%. This would not be possible with current tools that require users to have an in-dept knowledge of data ontology and advanced querying skills. Further, all the participants were highly satisfied (usability and acceptance) with our approach.

This work has raised interesting issues for future works. One relevant extension is to add modules and mechanisms to include software measurements that handle source code analysis rules as those used by PMD, Checkstyle, and FindBugs. Another interesting future work is to monitor the evolution of these measurements in order to support various change requirements. On the one hand, this means

adding a delta fact extraction feature that augments our current graph model with time sensitive data. On the other hand, end-users should have a means to filter data produced by each one of our workflow modules.

Another future work that has risen with the results obtained here is to conduct a full-scale comparative study that includes professional software analysts. We intend to implement the preliminary study recommendations and evaluate the performance of our prototype in comparison to existing software analysis tools on a corpus of software systems with a wider variety of software measurements.

3.A Appendix

3.A.1 Partial Listing of Implemented Fact-Extractors

	Name	BusinessLogic	Options
Project	AllProjects	All projects in the database	*Access:* INTERNAL, EXTERNAL, or BOTH
	CompilationUnitsInProject	All compilation units of a project	*Access:* INTERNAL only
	PackagesInProject	All packages of a project	*Access:* INTERNAL, EXTERNAL, or BOTH
Package	CompilationUnitsOfPackage	Compilation units of a package	*Access:* INTERNAL only
	ImmediateChildPackages	Immediate child packages of a given package	N/A
	PackageAggregation	Package subtree of a given package	N/A
	ParentPackage	Parent package of a given package	N/A
CU	AllTypesOfCompilationUnit	All Types declared in a compilation unit	*Access:* INTERNAL only **Note:** Alternatively use workflow to choose types
	ImportsOfCompilationUnit	All imported types of a compilation unit	*Access:* INTERNAL, EXTERNAL, or BOTH
	TopLevelTypeOfCompilationUnit	Top-level type of a compilation unit	*Access:* INTERNAL only
Type	AllTypesOfType	All Types declared in a type	**Note:** Alternatively use workflow to choose types
	AnonymousTypeDeclarationsOfStaticBlock	All AnonymousTypes declared in a static block	*Depth:* infiniteStatementDepth or statementDepth
	DeclaringCompilationUnit	Compilation unit in which the type was declared	*Access:* INTERNAL only
	DeclaringPackage	Package in which the type was declared	*Access:* INTERNAL, EXTERNAL, or BOTH
	DeclaringProject	Project in which the type was declared	*Access:* INTERNAL, EXTERNAL, or BOTH

Table 3.A.1. Description of Software Fact-Extractors

Table 3.A.1. Description of Software Fact-Extractors – continued

	Name	BusinessLogic	Options
Type	DepthInInheritanceTree	Find longest path to a root of the inherited tree for a given type	N/A
	FieldAccessFilterForType	Given a type and an accessed field, apply filter	*Access:* INTERNAL, EXTERNAL, or BOTH *:* sameCompilationUnitAllowed
	FieldsOfType	Field declarations of a type which may include inherited fields	*Depth:* infiniteInheritanceDepth or inheritanceDepth
	ImmediateParentsOfType	Immediate parents of a type	N/A
	ImmediateSubclassesOfType	Immediate subclasses (children) of a type	N/A
	InheritanceFilterForType	Given a type and a method, filter out local or inherited methods	*Filter:* localOnly or inheritedOnly
	MethodAccessFilterForType	Given a type and an accessed method, apply a filter	*Access:* INTERNAL, EXTERNAL, or BOTH *:* sameCompilationUnitAllowed
	MethodsOfType	Method declarations of a type which may include inherited methods	*Depth:* infiniteInheritanceDepth or inheritanceDepth
	NestedTypeOfType	Nested type declarations within a type	*Depth:* infiniteDepth or depth
	StaticBlocksOfType	All static blocks	N/A
	TopLevelStmtsOfStaticBlock	All top-level statements of a static block	N/A
	TypeAccessFilterForType	Given two types, an accessed and an accessor type, apply a filter	*Access:* INTERNAL, EXTERNAL, or BOTH *:* sameCompilationUnitAllowed
	TypeDeclarationsOfStaticBlock	All top-level type declarations of a static block	*Depth:* infiniteStatementDepth or statementDepth
	TypeFilter	Apply a types filter	*Depth:* infiniteStatementDepth or statementDepth

3.A.2 User Satisfaction Questions

- **Performance Expectancy**
  i. *I found the VIMETRIK Visual Query as a useful method for querying about the software system*
  ii. *I think that VIMETRIK Visual Query approach makes it quick to complete the task of querying about the software system*
- **Effort Expectancy**
  i. *I found it clear and understandable how to build a query from combining nodes*
  ii. *I found it easy to compose new queries from the existing nodes*
- **Utility**; Productivity
  i. *The visual query process supported the execution of required tasks*
  ii. *Overall, I liked the idea of composing the queries from making a workflow through the built-in nodes and meta nodes*
- **Intuitiveness**; Affordance, Transparency, Memorability, and Perspicuity
  i. *It was easy to know the outputs at each stage of building the query*
  ii. *It was easy to find out the "wrong" place in the workflow in the case of building wrong workflow*
  iii. *I found it helpful to see the visualized form of query results*
  iv. *I found it easy to understand the meanings of nodes*
  v. *Overall, the system was easy to use*
- **Learnability**; Feedback
  i. *Overall, the system was easy to learn*
- **Personal Effect**; Novelty, Satisfaction, and Stress
  i. *Overall, I feel comfort with using the system as a future tool*
  ii. *Overall, I am satisfied with my progress in the experiment*
  iii. *Overall, I would like to recommend this system to other users*

3.A.3 Software Understanding Tasks –
Gray marked rows were excluded from the experiment

Nr.	Task	Classification
1.1	Access all the packages of the Tomcat system. What is the total number of packages?	lightweight
1.2	Access all the compilation units (CU) of the system from the packages found. What is the total number of compilation units in the Tomcat system?	lightweight
1.3	Access all the types of the system from the compilation units found. What is the total number of types in the Tomcat system?	lightweight
1.4	Access all the methods of the system from the types found. What is the total number of methods in the Tomcat system?	lightweight
2.1	Access all the statements from the methods found. This means getting the top-level statements first, then getting all the nested statements, and merging them together. List the top-three methods (methodIds) with the most number of statements	advanced
2.2	Access all the expressions from the statements found. List the top-three methods (methodIds) with the most number of expressions	lightweight
3.1	From the expressions found access calls to other methods and apply a Filter. List the top-three methods (methodIds) with the most number of method calls	intermediate
3.2	Using the statements and expressions found, calculate the cyclic complexity of the method. List the top-three methods (methodIds) with the highest cyclic complexity (CC)	advanced
3.3	From the compilation units found, calculate the efferent coupling for each compilation unit. List the top-three cuIds with the highest efferent coupling	intermediate
3.4	From the compilation units found, calculate the afferent coupling for each compilation unit. List the top-three cuIds with the highest afferent coupling	intermediate
4.1	Preparing the Data. First step is that you are required to prepare the data. Follow what you have learnt so far to produce a table that contains the columns <projectId, packageId, cuId, typeId, methodId, Count(methodAccessId), CC>	advanced
4.2	Visualize the results using the CustomNetworkViewer and see if you can decipher the packages with the most significant CC	intermediate

Table 3.A.2. Software measurement experiment tasks

4

Visual Analysis of Software Architecture Evolution

" *I tell you and you forget. I show you and you remember. I involve you and you understand* "

– Confucius [1]

In Chapter 3, we looked into how visual analytics can help to specify and visualize user-defined software system measurements. In this chapter, we focus on the analysis of software architecture evolution. An essential component in the evolution and maintenance of large-scale software systems is to track the structure of a software system to explain how a system has evolved to its present state and to predict its future development. Current mainstream tools facilitating the structural evolution of software architecture by visualization are confined with easy to integrate visualization techniques such as node-link diagrams, while more applicable solutions have been proposed in academic research. To bridge this gap, we have incorporated additional views to a conventional tool that integrates an interactive evolving city layout and a combination of charts. However, due to a limited access to the stakeholders it was not possible to solicit them for a formal modeling process. Instead, an early prototype was developed and a controlled experiment was conducted to illustrate the vital role of such in-situ visualization techniques when aiming to understanding the evolution of software architecture.

[1] Was an influential Chinese philosopher, teacher, and political figure

This chapter is organized as follows. Motivation and context are provided in Section 4.1, the eCITY tool is described in Section 4.2, and the experiment setup and results are reported in Sections 4.3 and 4.4. Finally, we conclude this chapter and look at possible future work in Section 4.5.

4.1 Motivation

Mainstream software systems undergo continuous changes in order to adapt to new technologies, to meet new requirements, and to repair errors [151]. Inevitably, the software in question expands in both size and complexity, often leading to a situation where the original design gradually decays unless proper maintenance is performed [50]. However, due to the "complex, abstract, and difficult to observe" nature of software systems performing visually supported maintenance can be quite complicated [43]. The field of software visualization aims to ease this task by providing visual representations and techniques that make the software more comprehensible. A key ingredient of these visualizations is a visual representation of the software structure that assists in creating a mental map of the system. Such a mental map provides a means to examine product properties such as size and quality indicators and process events such as errors found or changes made [51].

With respect to the analysis of the evolution of software, it is essential to track the structure of the software system to explain and document how a system has evolved to its present state and to predict its future development [46]. There are a number of free and commercial tools that can be found in both academic and industrial research with the sole purpose of improving software architecture evolution comprehension through the use of visualization. On the one hand, industrial applications are confined to easy to integrate visualization methods and metaphors that lack the sophistication to handle informative large-scale software architecture evolution visualization. While on the other, academic researchers have developed numerous solutions that have not made it to the mainstream [44, 45, 191]. The work of Telea et al. [45] indicates two inter-related reasons for this phenomenon and we agree with their findings; 1) stakeholders do not have the time to try new tools to see if it fits in their context, and 2) tool developers cannot create an product that satisfies all possible

needs. Our aim is to bridge this gap by addressing the following two factors: 1) propose visualization methods and metaphors that do not significantly deviate from current solutions – analysts should be comfortable with the software architecture representations visually, and 2) the ability to monitor, visualize, and interact with large-scale software systems in real-time – be able to deal with the scale and complexity of real-world software applications.

The goal of this work is to utilize research ideas in the area of software architecture evolution visualization and to apply these modern techniques in the context of mainstream software architecture maintenance and evolution tools. To achieve this goal we have been working together with the Fraunhofer IESE, to enhance their conventional Software Architecture Visualization and Evaluation (SAVE) tool that evaluates software architectures while they are constructed as well as after their construction [192]. However, due to a limited access to the stakeholders it was not possible to solicit them for a formal modeling process. Instead, a prototype was developed to augment SAVE through the use of different views to further the daunting task of large-scale software architecture evolution analysis. While the experimental results show that some details were missed through this non-formal approach, they also show that an improved configuration of the visualization influences the efficiency and effectiveness of basic software architecture evolution tasks significantly. More specifically, a gain of efficiency by 170% and a gain of effectiveness by 15% in these basic tasks were realized simply by selecting a different set of views. Based on these results we claim that considerable benefits can be attained by incorporating such in-situ visualization methods and metaphors to conventional software architecture maintenance and evolution tools.

The resulting tool, eCITY[2], exploits all the benefits of the SAVE core functionality to generate Reflexion models [193] and its ability to conduct static architecture evaluations. In addition to the core SAVE visualization engine, views such as the *Timeline View* and the *City View* are incorporated to assist in the tracking of hierarchical structural changes over time.

[2] eCITY demo link: http://hci.uni-kl.de/~khan/eCITY/screencast.html

4.2 Methodology

In order to incorporate additional views to a conventional software maintenance tool, we designed our eCITY tool as an eclipse plug-in. We follow the well-known methodology of CMV and provide different architectural views of a large and complex software system. Thus, providing analysts with a means to "brush and link" elements for the purpose of evolution tasks. In this section, we provide details of the eCITY schema and the different views we provide.

4.2.1 eCITY Schema

In our work, we enhanced the Frauhofer SAVE tool by augmenting the architects' current workflow that relies solely on the SAVE Diagram View (see Section 4.2.2) - a view that provides extensive evaluation possibilities but one that does not have the best design to handle the complexities of exploring the systems evolution. To address these shortcoming, we provide additional linked views of the underlying data.

As Figure 4.1 shows, eCITY exploits the core functionality provided by the SAVE framework. In his thesis, Naab [194], examines this framework in more detail where as we focus more on the eCITY tool and the SAVE Diagram View. A key feature of the SAVE framework is responsible for extracting an architectural model from the source code of a software system. This model is generated through a combination of an initial fact extraction and a number of delta fact extractions, to produce data models that contain the target systems' architectural structure and its historical data. The eCITY tool interfaces with these models through a model interface (a generic JGraphT[3] graph implementation) to provide evolutionary software architecture visualizations; namely the *SAVE Diagram View*, the *Timeline View*, and the *City View*.

Although the SAVE Diagram View is part of the original SAVE tool, it was an important design decision to incorporate it in the eCITY tool so that the users have access to their original workflows. Both the Timeline and City Views have been designed to help the architects

[3] JGraphT: a generic Java graph library that provides mathematical graph theory objects and algorithms (https://github.com/jgrapht/jgrapht)

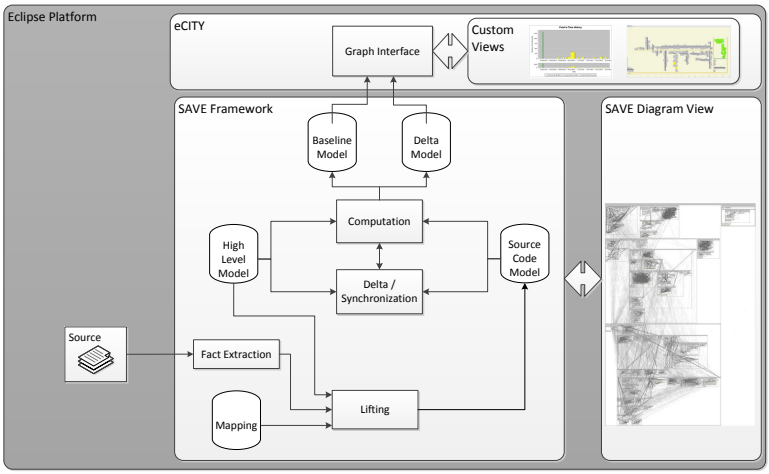
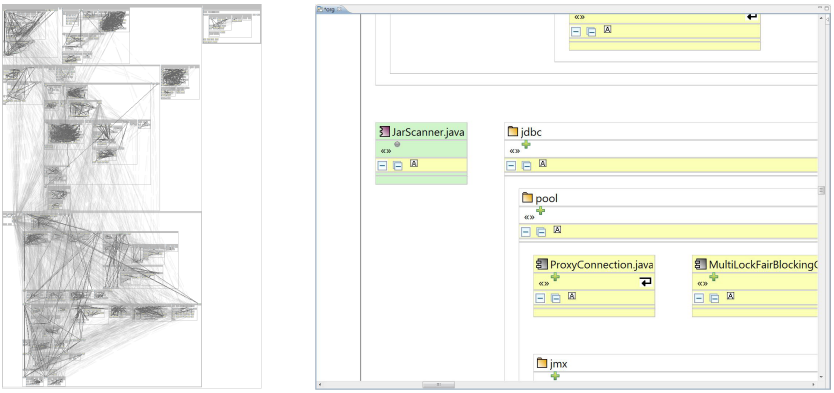


Figure 4.1. eCITY prototype schema

in accessing the structural changes of the software system in a more effective and efficient manner than using the SAVE Diagram View alone. We support this claim through the results of our experiment that are presented in Section 4.3.

4.2.2 SAVE Diagram View

The SAVE Diagram View (Figure 4.2a) is the main view used by the software architects at the Fraunhofer IESE to explore and assess a software systems architecture. The main features of this view are: its configurability (enabling and/or disabling certain graphical elements), the expressiveness of its graphical elements, and its rich-features that allow for extensive evaluations. Projects, packages, and classes are all represented as components in the SAVE Diagram View, while edges between components depict their relationships to one another. In general, SAVE has a nested approach in which high-level components have rectangular representations that may be either expanded one-level deeper into the hierarchy or expanded completely to show the entire underlying hierarchy. Conversely, components may be collapsed in a similar manner. The work of Knodel et al. [192] discusses these elements and their features in further detail.

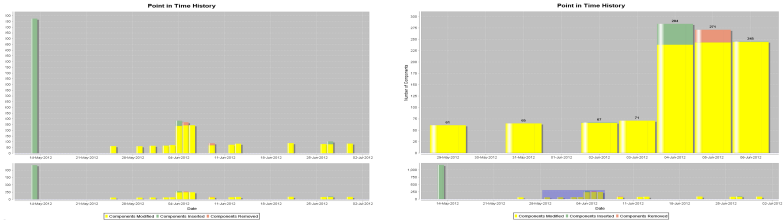


(a) Complete Diagram (b) Applying filters and scrolling

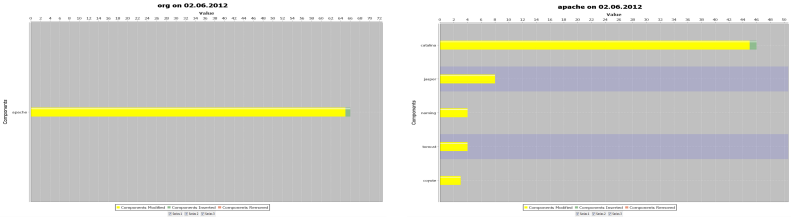
Figure 4.2. SAVE Diagram View

The complexity of a structural diagram is typically reduced by either creating new diagrams of selected components or through the use of various filters. For the purpose of analyzing the underlying software structure over time, the analysts rely on two distinct filters: the *Relation Type* filter and the *Point-In-Time* diagram filter. The user may reduce visual clutter, while solely examining the software structure, by applying a *Relation Type* filter to hide some or all of the displayed dependencies. Additionally, the user may apply a *Point-In-Time* diagram filter to explore the system at a particular point in time. As shown in Figure 4.2b applying a *Point-In-Time* diagram filter updates the SAVE Diagram View with icons to depict the modification status at a certain point in time according to the chosen parameters: a triangle depicts modifications, addition symbol depicts insertions, minus symbol depicts removals, and a circle depicts no change.

It was a conscious decision to incorporate the original view into the eCITY tool, thereby not completely replacing the architects' normal workflow. Instead, we provide them with the views described in Sections 4.2.3 and 4.2.4 to have additional perspectives of their evolution data.



(a) Timeline overview and detail plots



(b) Top components and org.apache on 02.06.2012

Figure 4.3. Timeline View

4.2.3 Timeline View

In the original SAVE workflow, architects apply a *Point-In-Time* filter to update the SAVE Diagram and manually track the number and type of changes made - a process which is deemed not only tedious but also error prone. The first step of this workflow requires the user to load the underlying data model that represents the software system being analyzed. It is during this process that data regarding the modification status of both the individual components as well as the overall hierarchy is stored; i.e. the distribution of these changes over time are stored in a convenient and easy to access manner.

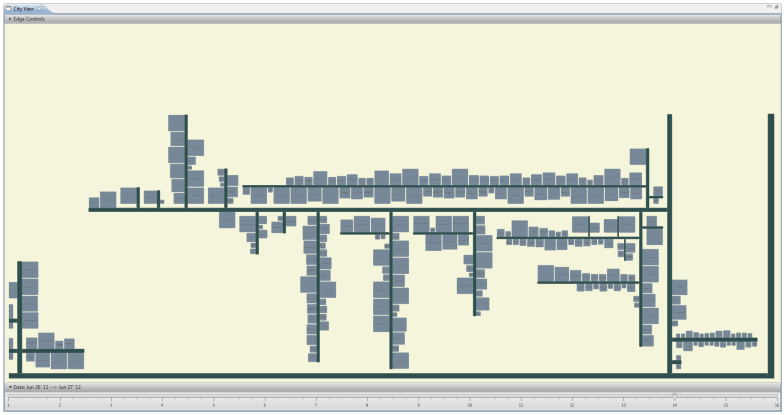
As such, the main purpose of the Timeline View is to provide the user with an overview of changes made to the system over time. A combination of interactive bar charts that represent the number of modifications, insertions, and removals made to packages and classes are employed to achieve this task. A typical color scheme was employed to depict these changes; modifications, insertions, and removals are represented using yellow, green, and red colors respectively.

The initial view consists of a combined plot (Figure 4.3a left); one of which is an overview where the user can select and manipulate a rectangular region, the other provides details on the selection (Figure 4.3a right). Further, the user may interactively select a point in time and update all three views simultaneously. Such a selection, changes the plot and provides details of the top level component at the chosen time stamp (Figure 4.3b left). Further, the user may recursively explore the distribution of changes over the hierarchy (Figure 4.3b right). The user may also navigate back to the previous chart or directly to the overview chart of Figure 4.3a. There are additional interaction possibilities with the charts; hovering over a chart component highlights the relevant subtree in the City View and selecting the chart component zooms onto the graphical representation of that particular component in the City View.

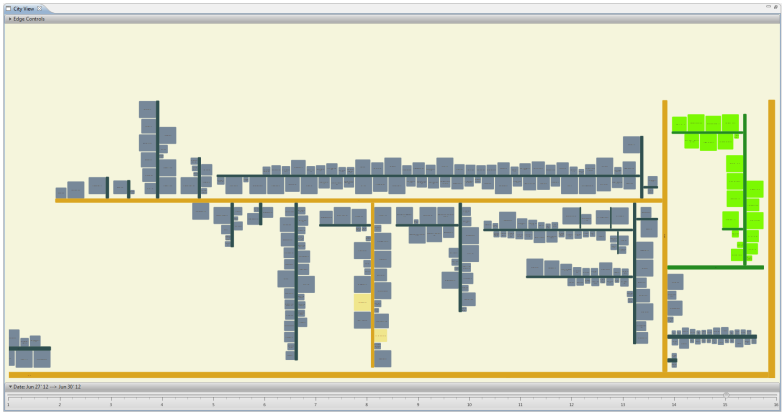
4.2.4 City View

An additional mode of analysis while monitoring the structural changes made to a software system is to track where these changes are made, i.e. where are packages and classes added, modified, or removed over time. In order to achieve this goal in the original SAVE workflow, architects apply multiple *Point-In-Time* filters and have to keep a mental map of these changes - this process is also deemed tedious, error prone, and depending on the size of the system hierarchy quite difficult. The City View (see Figures 4.4 and 4.5) addresses these concerns by providing an overview of the entire system architecture at a particular point in its evolution process and provides the user a means to interactively explore these changes over the system hierarchy.

It was mentioned in related works that our approach is inspired by the work of Steinbrückner et al. [51] and in particular the secondary model they present. While they focus on geometrically mapping different details of the development history onto this model, we choose to manipulate this layout to highlight structural changes over time. Figure 4.4a depicts our implementation of their secondary model, where packages are represented by streets and classes are symbolized by plots. The vertical or horizontal orientation of a street alters depending on its parent street/package orientation. Plots are positioned along the street representing the package they belong to; to reduce space requirements, they are positioned on both sides of the street. The layout algorithm adjusts the length of the street to hold all its



(a) Package org.apache.tomcat on 26.06.2012



(b) Package org.apache.tomcat on 27.06.2012

Figure 4.4. CITY View

plots and all its subpackages. Further, the size of a plot may vary according to a given scaling parameter; because connectivity is an important measure for an architect, we scaled plots according to its connectivity to other artifacts.

While the user loads the SAVE data model, we calculate the initial layout as described above and sequentially go through the *Point-In-Times* to update the layout by adding and removing both classes and packages. Such details of the city layout as well as the modification status of its components are stored in key frames to allow for real-time interaction. Utilizing a slider and key frame animation techniques colors are interpolated and the city suburbs grow and shrink to represent changes made between two points in time.

The city may be explored using the mouse to pan and zoom to examine its suburbs or different parts more closely. Further, the user may interactively invoke a slider and manually update the city to another point in time. This interaction mechanism animates the city through the use of colors that depict the modification status and the growing and shrinking of suburbs to represent the addition and/or removal of classes and package. This functionality may be observed by comparing Figure 4.4a with Figure 4.4b, where certain classes are modified (color changes to yellow) and a package is inserted in the top right corner (colored green). The color scheme employed is quite similar to that of the Timeline View charts, where yellow, green, red, and grey are used to depict the modification, insertion, removal, and unchanged modification status of each component.

Besides panning and zooming capabilities, two distinct modes of operation have been incorporated into this view; one of these modes is triggered when the user selects a point in time in the combined plot of Figure 4.3a, while the other mode requires the user to interact with a time slider. In the former mode the city may either animate or instantaneously update itself to the chosen point in time depending on whether the *Time Animation* is enabled or disabled. On the other hand, interacting with the time slider interactively updates the city to an older or a newer version depending on the direction chosen. As soon as the slider is released, the other two views are updated to reflect the active point in time. The user also has the possibility to reset the views or toggle certain animations using the pop-up menu. Finally, with mouse-over he is able to see the name of the underlying component, which comes in handy when he is zoomed out of the city.

Figure 4.5. Detailed eCITY View (rotated) – Left: Classes modified and Right: Package inserted

4.3 Experiment

As described in Section 4.2.2, architects apply filters to update the SAVE Diagram View to a particular point in time through the use of insertion, removal, and modification icons. However, it is quite difficult to keep a mental map of these changes due to the fact that this view uses nested components for displaying the containment of components - a view that tends to both dense and quite large. For improving the analysis of software architecture evolution over time, we proposed the Timeline and City views. We assume that through the use of these views, architectural changes would be tracked in a more efficient and effective manner. Furthermore, we assume that these views will be more useful than the current SAVE Diagram View. We tested this assumption by designing and performing a controlled experiment which compares the Timeline and City views (eCITY configuration) with the SAVE Diagram View (SAVE configuration).

4.3.1 Research Purpose and Hypotheses

The first goal was to compare the SAVE and eCITY configurations with respect to the efficiency and effectiveness achieved by analysts when they analyze architectural changes. Thus, we defined the following research hypotheses:

- H_1: Analysts using the eCITY configuration are more efficient than analysts using the SAVE configuration when they analyze the evolution of software architectures.
- H_2: Analysts using the eCITY configuration are more effective than analysts using the SAVE configuration when they analyze the evolution of software architectures.

The second goal was to compare SAVE and eCITY with respect to their acceptance and usability. Therefore, we define the two additional research hypotheses:

- H_3: Analysts accept the eCITY configuration more than the SAVE configuration when they analyze the evolution of software architectures.
- H_4: Analysts consider the eCITY configuration more useful than the SAVE configuration when they analyze the evolution of software architectures.

4.3.2 Operationalization

In order to test the above hypotheses, we operationalized the four variables of interest, selected a software system to be analyzed, and designated the tasks to be performed. First, the variables of interest were operationalized as follows:

i. Efficiency is the time required for accomplishing a set of given tasks.
ii. Effectiveness is the difference between the true and actual score related to a task.
iii. Acceptability is measured using the TAM, which is a valid and reliable questionnaire for assessing technology acceptance and use [189]. Out of 31 questions in the original TAM, we selected 7 questions focused on performance and effort expectancy (see Appendix 4.A.1). All questions were rated using a five-point Likert scale (1: I strongly disagree, 5: I strongly agree).
iv. Usability is measured using the questionnaire proposed by Nestler et al. [190]. Considering the purposes of the eCITY configuration, we selected 4 out of 5 defined dimensions and 15 out of 269 questions. The selection was discussed with evaluation experts. The dimensions and related questions are listed in Appendix 4.A.2. Each question is rated using a five-point Likert scale (1: I strongly disagree, 5: I strongly agree).

Second, we selected the Apache Tomcat system as the system to investigate because it is a real software system and its architectural models were available in SAVE. Finally, three types of architectural evolution tasks were identified with the support of experts. These are:

i. *Counting:* Identifying changes made to the system on a specific date or changes made to a subcomponent in a time period.
ii. *Find Date:* Identifying the time period with the most number of changes or when a component has been changed the most.
iii. *Find Package/s:* Identifying the subcomponents of a chosen component that have been changed in a time period, find modules / subcomponents that are present since the earliest version and have not been changed since, or find modules / subcomponents that have changed a lot in a particular time period.

4.3.3 Field Test

A field test was conducted with two experts in the field of software architecture. Our purpose was to get an early feedback regarding the experimental design, the selected tasks, and the time required for solving them.

The first expert completed the tasks using the SAVE configuration in an hour and fifteen minutes. He considered the tasks to be realistic. However, he suggested that while asking the user if certain components are changed often or not at all, we should focus on a smaller sub-package rather than the entire system. Hence reducing the time required to perform the tasks related to finding packages.

The second expert solved the tasks using the eCITY configuration in forty five minutes. He also considered the set of selected tasks to be realistic. Additionally, he made recommendations regarding the visualization. These included: 1) keeping the date format and color coding uniform, 2) adding the option to enable and disable animations, and 3) adding feedback regarding the enabled/disabled features in the City pop-up menu. These issues were discussed and solved.

Consequently, the original set of tasks were tweaked to address the experts' feedback.

4.3.4 Controlled Experiment

Subjects

The controlled experiment was conducted with 38 participants. Out of the 38 participants, 3 were software engineering experts of the Fraunhofer IESE and 35 were computer science graduate students from the Technical University of Kaiserslautern. The three experts involved have a deep knowledge in the field of software architecture and prior experience in SAVE, while all the students have theoretical knowledge about software architectures.

Due to the lack of experts available and their previous experience in SAVE, experts were asked to work with SAVE and then with eCITY. Students were randomly distributed in two groups working either with SAVE or eCITY. This was intended for avoiding learning effects.

Experimental Setup

The same software architecture was explored using either SAVE or eCITY. Thus, two computers were prepared for the experiment: one with Eclipse and the SAVE Configuration and the other with Eclipse and the eCITY configuration. The Eclipse workspace was prepared for both installations to contain the required data: a SAVE data model containing 16 fact extractions of the Apache Tomcat system between the time period 14.05.2012 and 02.07.2012. Each instance of the Apache Tomcat system was extracted from its public repository. Four of these instances were complemented with the insertion, modification, and removal of some fictitious packages and classes to create some interesting artifacts.

Each workspace was completely restored at the beginning of each experiment session. A different visualization introduction handout was provided to each participant depending on the installation used. Additionally, the participants were afforded with a walkthrough of the functionalities and features that would assist them in the experiment process. The participants were then asked to perform the tasks and questionnaires described in Section 4.3.2.

Each participant was allowed unlimited time to finish the tasks and to fill the questionnaires regarding Acceptability and Usability. The resulting materials were then collected and analyzed.

Data Collection

The time and the answer to each task was collected using an exercise sheet. Additionally, a printed questionnaire was used for eliciting the perception of the participants regarding the acceptance and the usability of the corresponding configurations, i.e. SAVE or eCITY.

Data Analysis

A transcript of the collected efficiency, effectiveness, acceptability, and usability data was compiled in excel. The subject data is kept anonymous and confidential. Regarding effectiveness, tasks were weighted according to their difficulty: Counting and Find Date tasks were awarded 3 points each and Find Package/s tasks were awarded 4 point each. The latter sets of tasks were weighted higher as they were deemed to be more complex.

We applied descriptive statistics methods such as the sample mean, standard deviation, median, and range to the experimental data. Further, we used the *MegaStat* excel plug-in to statistically test our hypotheses using the Mann-Whitney-U-Test. It is appropriate for our scenario due to the size of the data and its ability to handle both normal and non-normal distributions. We performed the tests with a confidence level for rejecting the null hypotheses at 99.9%.

4.4 Results

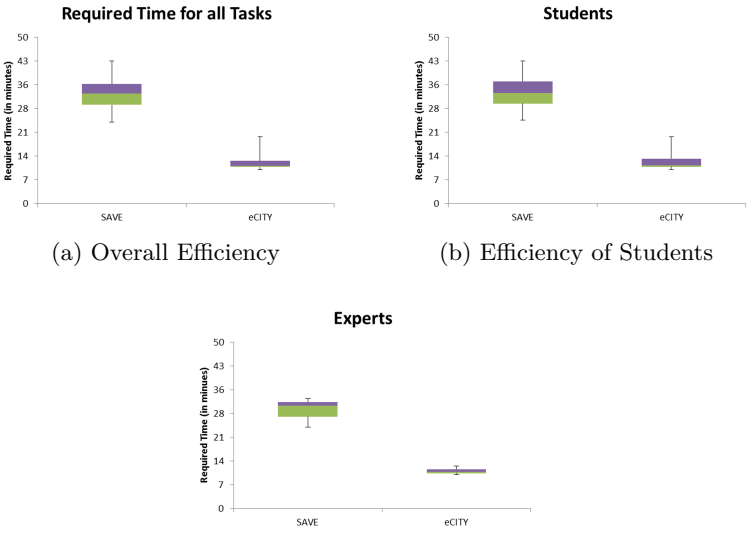
The statistical evaluation of the experiment is presented in this section. As such, the time required for tasks, the results of the tasks, the acceptability of the configuration, and the usability of each configuration is presented. Further, we argue for the earlier stated hypotheses by performing hypotheses testing on the collected data. We conclude this section by examining the threats to validity.

4.4.1 Efficiency Results

The time required for each participant to complete a task was measured to compare the efficiency with which the participants performed using each one of the two configurations. The results are presented in the box-plots of Figure 4.6; Figure 4.6a depicts the efficiency of the participants with respect to the SAVE and eCITY configurations, while figures Figure 4.6b and Figure 4.6c examine the efficiency of students and experts respectively. Based on these results, the following observations can be made:

- Independent from being experts or students, members of the eCITY group were on average more efficient.
- There is a larger variance from the average required time in the SAVE group than the eCITY group.
- Independent from the configuration used, students were on average as efficient as experts.
- The variance of the results is larger for students than experts

The mean, median, and standard deviation values of the efficiency with which the participants performed are presented in Table 4.1. The results presented have been broken down to the type of tasks, to inspect the level of efficiency reached for each type. It is not



(a) Overall Efficiency

(b) Efficiency of Students

(c) Efficiency of Experts

Figure 4.6. Box-plots of Efficiency results

surprising to see such a significant improvement in the efficiency of counting tasks as the SAVE configuration required the participant to pan and zoom on an extremely large viewing area and physically count the changes while the eCITY configuration required them to look up charts and interact with sliders. The participants needed to interact with the eCITY configuration in much the same manner for the Find Date tasks, however, for the SAVE configuration they had

		Mean	Median	Std. Deviation
Counting	SAVE	11:20	11:15	2:11
	eCITY	1:59	1:58	0:32
Find Date	SAVE	8:09	7:29	2:46
	eCITY	2:12	2:07	0:42
Find Package/s	SAVE	13:50	13:19	2:11
	eCITY	8:10	7:50	1:39

Table 4.1. Participants Efficiency (time required in mm:ss)

to apply filters and compare the densities of each updated diagram. The biggest challenge they faced was having to keep a mental map of these changes. Similarly, the participants were more efficient in the Find Package/s tasks using the eCITY configuration as they found it easier to interactively update the city and locate both stable and heavily constructed areas.

In Table 4.2, we further examine the results from the SAVE (A) and eCITY (B) configurations. Using this table we can compare the efficiency gain of the eCITY configuration to that of the SAVE configuration. The following efficiency gains were recorded: 470% for counting tasks, 271% for finding significant dates, and 69% for finding packages with certain changes. This equates to an overall efficiency gain of 170% for the eCITY configuration compared to the SAVE configuration. Knodel et al. [195] conduct a similar empirical experiment where they also evaluate the results with respect to the effect size, a representation of the difference in mean values as compared to the standard deviation. Similarly, we calculate the standard deviation in Table 4.2 using the formula of Hedges et al. [196] and claim an overall effect size of 5.77 standard deviations to be highly significant [197].

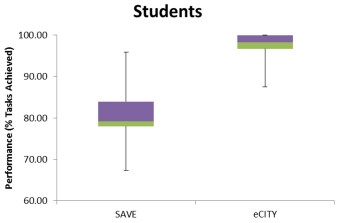
	Mean A	Mean B	Dev.	Gain	Effect
Counting	11:20	1:59	1:20	469.91	6.99
Find Date	8:09	2:12	1:42	270.78	3.49
Find Package/s	13:50	8:10	1:55	69.37	2.97
Overall	33:18	12:21	3:38	169.65	5.77

Table 4.2. Efficiency gain and effect size (mm:ss)

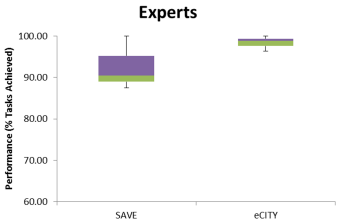
The corresponding efficiency hypothesis was tested using the Mann-Whitney-U-Test. These results indicate that on average the eCITY configuration is significantly more efficient than the SAVE configuration.

$H_{0,1}$: $\text{Time}_{\text{SAVE}} <= \text{Time}_{\text{eCITY}}$
Test input: the time required to complete tasks
Test: Mann-Whitney U test (one-tailed)
Result: Z=5.46, p=2.32E-8, $p < 0.001$; thus, the null hypothesis is rejected

(a) Overall Effectiveness



(b) Effectiveness of Students



(c) Effectiveness of Experts

Figure 4.7. Box-plots of Effectiveness results

4.4.2 Effectiveness Results

The accuracy with which each participant completed a task was measured to compare the effectiveness with which the participants performed using either one of the two configurations. As such, the answers to the tasks were evaluated against the expected results. The maximum score was 24 points (6 points for Counting tasks, 6 points for Find Date tasks, and 12 points for Find Package/s tasks). The results are presented in the box-plots of Figure 4.7. A similar pattern to the efficiency evaluation is followed: Figure 4.7a depicts the effectiveness of the participants with respect to the SAVE and eCITY configurations, while figures Figure 4.7b and Figure 4.7c examine the effectiveness of students and experts respectively. The following observations can be made from the results presented:

- Independent from being experts or students, members of the eCITY group were on average more effective.
- There is a larger variance from the average value in the SAVE group than the eCITY group.

- Experts were on average more effective than students using the SAVE configuration, however, the students performed almost as well as experts with the eCITY configuration.
- The variance of the results is larger for students than experts.

The mean, median, and standard deviation values of the effectiveness with which the participants performed are presented in Table 4.3. Again, to examine the level of effectiveness reached for each type of tasks, the results have been broken down for each type. Our findings show that panning and zooming on a large view to count the changes leads to more errors than using the chart and slider combination of the eCITY configuration. It was difficult for the participants to keep track of which components they had already counted. The participants also made more mistakes with the SAVE configuration while looking for dates with the most changes as they found it difficult to compare the mental maps of the densities of each updated diagram. It was hard for them to distinguish between the relatively few changes between two different time stamps. For the last category of tasks, the participants performed equally well to find certain packages that were modified in a particular time frame. However, there was a larger variance from the average effectiveness using the SAVE configuration as compare to the eCITY configuration.

		Mean	Median	Std. Deviation
Counting	SAVE	70.00	66.67	16.75
	eCITY	99.21	100.00	3.64
Find Date	SAVE	70.00	50.00	29.91
	eCITY	97.62	100.00	10.91
Find Package/s	SAVE	95.95	97.62	5.34
	eCITY	96.94	97.62	3.03

Table 4.3. Participants Effectiveness (% of task results achieved)

We further examine the effectiveness results from the SAVE (A) and eCITY (B) configurations in Table 4.4. The following effectiveness gains were recorded: 29% for counting tasks, 28% for finding significant dates, and 1% for finding packages with certain changes. This equates

to an overall effectiveness gain of 15% for the eCITY configuration compared to the SAVE configuration. The effect size for all tasks is 2.52 standard deviations, which according to Cohen [197] is considered to be a large effect size.

	Mean A	Mean B	Dev.	Gain	Effect
Counting	70.00	99.21	10.04	29.44	2.91
Find Date	70.00	97.62	20.18	28.29	1.37
Find Package/s	95.95	96.94	4.16	1.02	0.24
Overall	82.98	97.68	5.83	15.05	2.52

Table 4.4. Effectiveness gain and effect size (%)

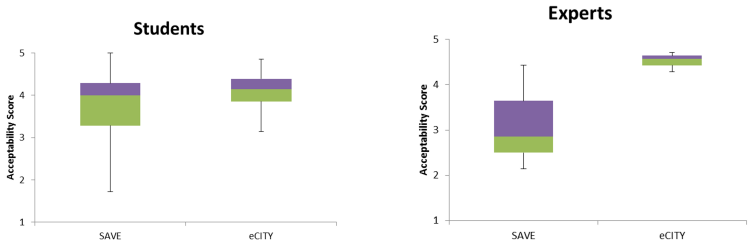
The corresponding effectiveness hypothesis was tested using the Mann-Whitney-U-Test. These results indicate that on average the eCITY configuration is significantly more effective than the SAVE configuration.

$H_{0,2}$: $\text{Effectiveness}_{\text{SAVE}} >= \text{Effectiveness}_{\text{eCITY}}$
Test input: the effectivess (accuracy) of the task results
Test: Mann-Whitney U test (one-tailed)
Result: Z=-4.81, p=7.55E-7, $p < 0.001$; thus, the null hypothesis is rejected

4.4.3 Acceptability Results

The participants of the experiment were asked to give their personal assessment on the acceptability of the two configurations. As mentioned in Section 4.3.2, we applied certain parameters of the performance and effort expectancy dimensions of the Technology Acceptance Model. The participants responses ranged from 1 (Strongly Disagree) to 5 (Strongly Agree) for each parameter. In Table 4.5, we present the aggregated mean, median, and standard deviation values for the dimensions mentioned above and the average overall acceptability score. These results indicate that the average values in all of these cases are very similar for both configurations.

Using the box-plots of Figure 4.8, we can make the following observations regarding the acceptability scores for each configuration by participant type:



(a) Students Acceptability Score (b) Experts Acceptability Score

Figure 4.8. Box-plots of Acceptability response

- The average acceptability score for the experts differs more than the score of the students.
- Independent from being experts or students, the acceptability score of the eCITY group was higher on average.
- Students voted higher for the acceptability of the SAVE configuration than experts, whereas experts voted higher still for the eCITY configuration.
- The difference in expert acceptability scores for the two configurations is higher than the difference in student acceptability scores.

At first the last observation was a bit surprising, however, after careful consideration it was in fact quite a reasonable result. We claim this to be the case due to the fact that the student evaluations were blind, that each student used only one configuration, and most importantly the experts were more aware of the shortcomings in the SAVE configuration.

		Mean	Median	Std. Deviation
Performance E.	SAVE	3.72	4.33	1.20
	eCITY	4.38	4.33	0.43
Effort E.	SAVE	3.56	3.75	0.86
	eCITY	3.94	4.00	0.52
Overall	SAVE	3.63	3.93	0.94
	eCITY	4.13	4.14	0.42

Table 4.5. Participants Acceptability response

The corresponding acceptability hypothesis was tested using the Mann-Whitney-U-Test. These results indicate that on average the eCITY configuration achieves the same acceptability as SAVE configuration.

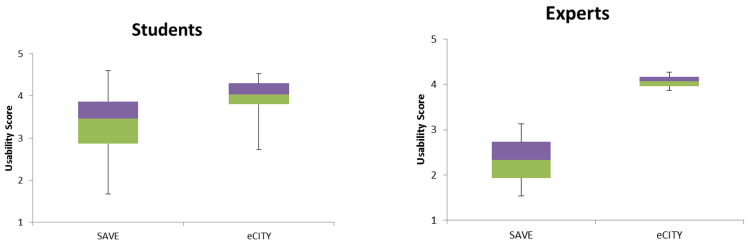
$H_{0,3}$: $\text{Acceptability}_{\text{SAVE}} >= \text{Acceptability}_{\text{eCITY}}$
Test input: results of the acceptability questionnaire
Test: Mann-Whitney U test (one-tailed)
Result: Z=-1.55, p=0.0608; $p > 0.001$; thus, the null hypothesis cannot be rejected

4.4.4 Usability Results

The participants of the experiment were also questioned for an assessment on the usability of the two configurations. Earlier in Section 4.3.2, we defined Usability in terms of various Utility, Intuitiveness, Learnability, and Personal Effect parameters. Usability was evaluated in the same manner as Acceptability where the participants used an ordinal scale with five values for their responses. The aggregated mean, median, and standard deviation values for the dimensions mentioned above and the average overall usability score are presented in Table 4.6. These results indicate that the average values in three of the four dimensions are marginally higher for the eCITY configuration, while the average value in the fourth dimension (Personal Effect) is

		Mean	Median	Std. Deviation
Utility	SAVE	3.48	3.67	0.97
	eCITY	4.22	4.00	0.53
Intuitiveness	SAVE	3.29	3.75	0.84
	eCITY	3.81	4.00	0.61
Leanability	SAVE	3.30	4.00	1.22
	eCITY	4.14	4.14	0.65
Personal Effect	SAVE	3.02	2.90	0.94
	eCITY	4.03	4.20	0.43
Overall	SAVE	3.24	3.30	0.79
	eCITY	3.99	4.07	0.40

Table 4.6. Participants Usability response



(a) Students Usability Score (b) Experts Usability Score

Figure 4.9. Box-plots of Usability response

significantly more. Overall, the average usability score of the SAVE configuration leans towards *neutral* on the scale, while the eCITY configuration tilts towards *agree* on the scale.

By examining Figure 4.9 we can make the following remarks about the usability scores for each configuration by participant type:

- The average usability score for the experts differs more than the score of the students.
- Independent from being experts or students, the usability score of the eCITY group was higher on average.
- On average, the students voted a whole ordinal scale higher for the usability of the SAVE configuration than experts.
- The difference in expert usability scores for the two configurations is higher than the difference in student usability scores.

The corresponding usability hypothesis was tested using the Mann-Whitney-U-Test. These results indicate that on average the eCITY configuration is perceived to be more useful than the SAVE configuration.

$H_{0,4}$: $\text{Usability}_{\text{SAVE}} >= \text{Usability}_{\text{eCITY}}$
Test input: results of the usability questionnaire
Test: Mann-Whitney U test (one-tailed)
Result: Z=-3.33, p=4.0E-4, $p < 0.001$; thus, the null hypothesis is rejected

4.4.5 Threats to Validity

Threats to internal and external validity are discussed in this section. Threats to internal validity refer to conditions that constrain the confidence level of the results, while threats to external validity are conditions that limit the generalization of the study.

Internal Validity

We tried to prevent the effects of confounding variables, between a tool (independent variable) and a dependent variable (efficiency, effectiveness, acceptability, and usability). Student participants received the same training for either the SAVE or eCITY configurations. None of the thirty five students had any prior experience with SAVE or eCITY and were randomly distributed in to one of the two study groups. We then ran the experiment with SAVE experts to make sure that the results were representative. The three experts had no experience with the eCITY configuration and were given the same training as the students. As our statistical results show, there was not a lot of difference in the dependent variable within each configuration. Further, we verified that the tasks for each configuration were performed with similar performance and time. The acceptability and usability aspects we choose are quite reliable, but certainly there are other measures. We applied the widely accepted Technology Acceptance Model and the usability study of Nestler et al. to break down the two aspects into 22 distinct measures. Lastly, we argue that the Apache Tomcat servlet container is representative of a real-world software system. It is used in a diverse range of industries and organizations to power numerous large-scale web applications.

External Validity

A real and complete system, the Apache Tomcat, was evaluated by the participants using the two configurations. Although, our results show that our solution was as acceptable as the original configuration, but more efficient, effective, and perceived as more useful, it would be interesting to repeat the experiment with other representative systems to check if the validity holds. Further, since a small number

of participants were involved, the experiment should be repeated with more participants. These two measures should ensure that our approach has a practical value for other projects that look to improve analysis of software evolution.

4.5 Concluding Remarks

The quality of a software system's architecture is one of the most significant factors for its success, a characteristic that is even more prevalent in the development of large and complex software systems. A precursor to accessing the quality of the underlying software architecture is to comprehend the structure of its elements. Further, the structure of the architecture continuously undergoes changes to adapt to functional or quality requirements, making the comprehensibility of the existing architecture an effort intensive activity. These problems are addressed through the use of visualization tools that examine the evolution of software architectures. The experiment we conducted showed the significance of employing appropriate visualization techniques and metaphors to conduct such analyses. By means of a more appropriate configuration, participants achieved an average gain of 170% in the efficiency and an average gain of 15% in the effectiveness in basic software architecture evolution tasks.

While the results indicate a higher acceptability and usability vote for our solution, we cannot statistically guarantee the former. The distribution of participants may be attributed to this observation; we had 35 students and 3 experts who took part in the experiment. Our results show that not only did students vote higher for the original configuration, but also that the average acceptability and usability expert votes differed more significantly for the two configurations than the student votes. We attribute this discrepancy to the students lack of practical experience in software architecture practices and to the experts being more aware of the shortcomings of the original configuration.

For the realization of our goals, we worked with Product Line Architects at the Fraunhofer IESE to augment their workflow with additional views of their data. In this chapter, we introduced an implementation of an interactive evolving city view through a non-formal modeling process that proved to be both efficient and effective.

Initially, the stakeholders were reluctant with the idea of a city layout as it deviated from traditional node-link layouts. However, they revised this opinion by working with the eCITY prototype and even found the interactive city to be both natural and intuitive.

By not following a formal modeling process there are certain details that we need to address further. The most critical limitation is that we currently do not examine how the interdependencies (relations) of a software architecture evolve over time. While the experts can still use their traditional workflows to examine these dependencies, "it would be quite nice to add them to the city view". Other possible improvements include a better mechanism for locating packages and classes such as an integrated search engine, highlighting of suburbs undergoing change in a time interval using multiple fish-eye views, the implementation of an alternate color scheme to address colorblindness, and an interface scheme that allows users to perceive and interact with focus and contextual views through overview-and-detail or focus-and-context. We are also aware that at times the analyst needs to directly compare the architecture at two disjoint points in time and will be addressing this in the near future.

Overall, the results show that our solution was in average as acceptable as the original configuration, but was more efficient, effective, and perceived as more useful. These results are a positive indication of the quality of our solution in terms of efficiency, effectiveness, acceptance, and usability. However, further empirical studies are required for confirming these results and deriving conclusive outcomes; i.e. using a larger sample, analyzing different systems, etc.

4.A Appendix

4.A.1 Acceptability Questions

- **Performance Expectancy**
  i. *I would find the visualization useful in my job*
  ii. *Using the visualization enables me to accomplish tasks more quickly*
  iii. *Using the visualization increases my productivity*
- **Effort Expectancy**
  i. *My interaction with the visualization would be clear and understandable*
  ii. *It would be easy for me to become skillful at using the visualization*
  iii. *I would find the visualization easy to use*
  iv. *Learning to apply the visualization is easy for me*

4.A.2 Usability Questions

- **Utility**; Productivity
  i. *I find the visualization highly appropriate to get details on the software's architectural evolution*
  ii. *The visualization supported the handling of all the tasks I needed to perform*
  iii. *I was highly successful in accomplishing the given tasks with the visualization*
- **Intuitiveness**; Affordance, Transparency, Memorability, and Perspicuity
  i. *The visualization clearly indicated all the possible inputs to me*
  ii. *I find the terms, abbreviations, and symbols used in the visualization easy to understand*
  iii. *I find the visualization to be highly understandable*
  iv. *I find the effects of actions to be highly transparent*
  v. *I find the visualization helped reduce my memory load in accomplishing the given tasks*
  vi. *It was easy for me to find the important commands and actions*
- **Learnability**; Feedback
  i. *The visualization provided appropriate feedbacks to me when I interacted with it*

- **Personal Effect**; Novelty, Satisfaction, and Stress
  i. *I feel that the visualization provided a novel approach*
  ii. *I was totally comfortable with using the visualization*
  iii. *I find interaction with the visualization to be pleasant*
  iv. *I am satisfied with the information provided to me by the visualization*
  v. *I felt insecure, discouraged, irritated, or stressed while using the visualization*

5

Visual Analysis of Software Architectural Relations

" *Simple things should be simple, complex things should be possible* "

– Alan Kay [1]

Current tools to analyze software architecture structure and its evolution tend to focus more towards exploring the architectural elements, their quality measures, and their evolution as compared to the information contained in the inter-dependencies of the system itself. In this chapter, the ideas presented in Chapter 4 are extended to provide an interactive visualization aimed at assisting software engineers and architects to additionally comprehend the architectural ties between software components. Further, the proposed methodology examines the evolution of these relations over time. To achieve these goals, the extended tool Evolving CITY-plus (eCITY+) [59] represents architectural relationships as hierarchical edge bundles on top of an evolving software city metaphor and unravels the evolution of these relations through the use of animations. Additionally, a discussion is presented on the challenges faced while implementing these ideas in the large.

[1] An American computer scientist known for his early pioneering work on object-oriented programming and windowing graphical user interface design

This chapter is organized as follows. Motivation and context are provided in Section 5.1. The proposed interactive visual approach of examining the architectural relationships using the eCITY+ tool is described in Section 5.2. In Section 5.3, insights gained by applying the proposed approach to examine a real software system are provided. Finally, we conclude this chapter in Section 5.4.

5.1 Motivation

Generally, reverse engineering aims at achieving two objectives: 1) procuring a mental model of a software system to comprehend its most recent version, and 2) examining the system history to explain how a system has evolved to its present state, repair errors, and to predict its future development [11, 151]. It is important in both cases to have a clear understanding of the software architecture – an understanding that includes the analysis of not only the architectural structure but also the relations between software elements. Further, while undertaking the daunting task of software evolution the complexity of the analysis increases due to the huge quantity of evolution data [49]. This poses interesting research questions as how to best handle the visual explorations in a meaningful and efficient manner.

There are a number of tools available that employ a combination of metaphors and techniques to analyze the hierarchical decompositions of software (e.g., Lattix, IBM Rational Architect, Bauhaus, Rigi, SHriMP, BugCrawler, etc.) [45]. However, only a few emphasize on the importance of the relations between software components. While these tools recover dependency graphs from the software system, they usually focus on nodes in the graph; as such, they contain only the static representations of the architectural elements' inter-dependencies [19]. As a result interdependencies are depicted using still images, providing the user with no direct means to examine the evolution of these relations.

In this chapter, we argue for a need to focus on both the static and evolution-based analyses of the inter-dependencies of software components to grasp more a comprehensive understanding of the software architecture than by just focusing on the hierarchical structure. To achieve this goal, we have developed the eCITY+ tool[2] (see Figure 5.1), which is an enhanced version of our previously developed

[2] eCITY+ demo link: http://hci.uni-kl.de/~khan/eCITY/screencast2.html

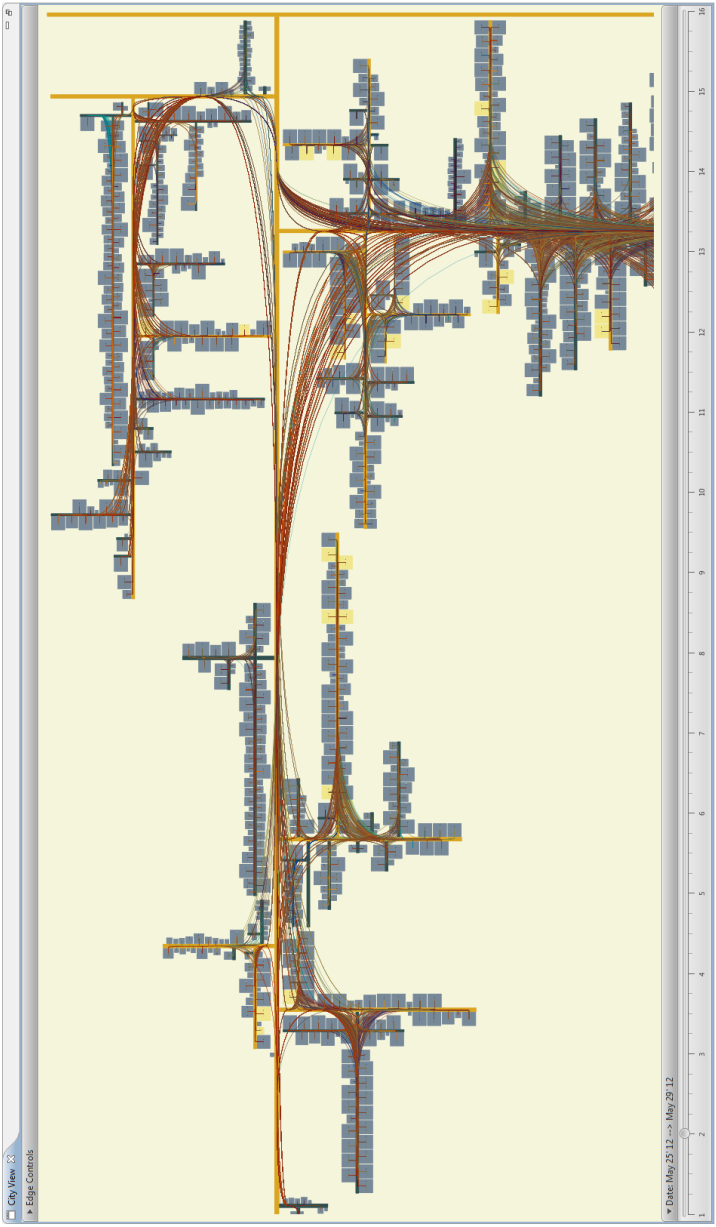


Figure 5.1. Analyzing relations with eCITY+

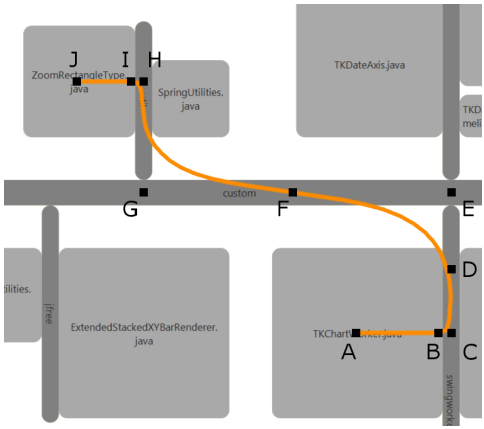
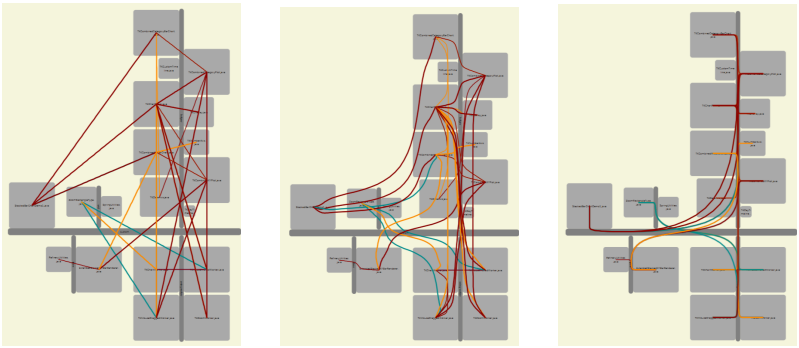


Figure 5.2. Creating an architectural edge

eCITY tool [57] that proposes stable city layouts for evolving software systems. Our tool is consciously designed as a plug-in to a traditional software maintenance tool (i.e., SAVE [192]). As such, the maintenance tool extracts historical recordings and interfaces with eCITY+ to have an additional view on the evolution of the software. The resulting interactive visualization represents the software hierarchy in the form of the popular city metaphor and overlays the remaining relations of the system components as HEBs [5]. Additionally, instead of using traditional means (e.g., arrows or color gradients) to show the directions of these relations, it interactively employs particle animations. Further, it animates both the hierarchical and inter-dependency representations to visually compare changes over a user-defined time period.

5.2 Methodology

In this section, we present the approach of our eCITY+ tool, an enhanced version of our previous built eCITY [57] tool, for representing architectural relationships as hierarchical edge bundles on top of the city layout. Additionally, we examine the different interaction possibilities.



(a) $\beta = 0.0$ (b) $\beta = 0.5$ (c) $\beta = 1.0$

Figure 5.3. Varying edge bundling factor

5.2.1 Overlaying Architectural Relationships

In the eCITY+ tool, we implemented the HEB technique to reduce the visual clutter when dealing with a large number of edges. Figure 5.2 depicts one such edge between two nodes, where the path along the hierarchy between these nodes marked with the letters A through J is used as the control polygon of a spline curve. This control polygon is constructed in a similar fashion to the one presented by Holten et al. [5], where it contains points along the hierarchy from the source node through the least common ancestor to the destination node (i.e. the path (A,C),(C,E),(E,G),(G,H), and (H,J)). However, in order to have a more aesthetically pleasing representation in the city layout, additional points (B, D, F, and I) are inserted in the control polygon to route each relation closer to the streets. In our case, users can interactively change the HEBs bundling factor β to straighten the spline curve (see Figure 5.3).

Additionally, in eCITY+ we implemented the Bellman-Ford shortest path algorithm to determine the path for each edge. While traversing this path the relative orientation of the intermediate nodes is checked to calculate the intersecting points. In the example of Figure 5.2, this yields the points C, E, G, and H. Points of the leaf node, such as A, B, I, and J are positioned at the center of the leaf nodes and

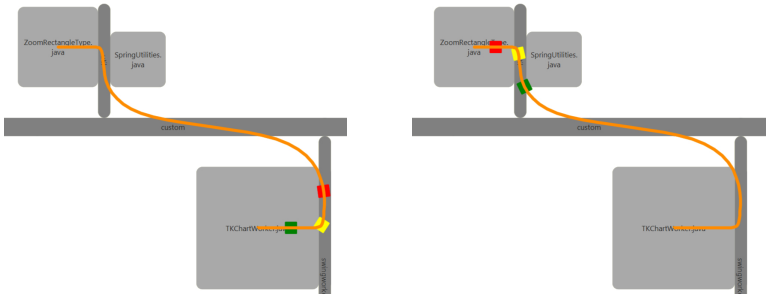
the center of the intersecting line segment between each leaf node and its parent. Further, points (D and F) are inserted in the middle of the line segments that join each subsequent pair of the above-mentioned intersecting points.

As shown in Figure 5.1, these bundled relations are mapped on top of a software city decomposition of the Apache Tomcat system. The figure is a depiction of the system on 25.05.2012 where streets and plots represent packages and classes and their colors are a representation of the modification status at this time stamp compared to the previous time stamp 14.05.2012 (grey, yellow, red, and green imply no-change, modification, insertion, and removal respectively). Further, the color of each adjacency edge represents the type of relationship (orange, red, green, yellow, cyan, blue, purple are used for access, call, dependency, dynamic invocation, import, inheritance, and interface relations respectively).

5.2.2 Particle Animations to Depict Direction

In traditional edge representations, curvature or arrows indicate the direction of the relation [5]. However, we could not use curvature as it is required to generate the edge bundles and arrows would clutter the visualization [5, 136]. Further, it may also not be ideal to use an RGB interpolated color gradient in a large system representation with many relations especially when color is used to depict the type of relation.

To address these issues we have implemented a particle animation that is triggered interactively. The idea is similar to how *runway-lights* are used in airports to land a plane. As shown in Figure 5.4, the user may select an edge to have these particles (red, yellow, and green rectangles) transition along the edge path from source (see Figure 5.4a) to destination (see Figure 5.4b). The direction of the relationship is indicated by the combination of the color arrangement of these particles and their path transition. This animation then runs in an infinite loop until the user stops it by selected the edge again. Further, the user may select multiple edges to have several such animations run simultaneously.



(a) Flowing from source (b) Flowing to destination

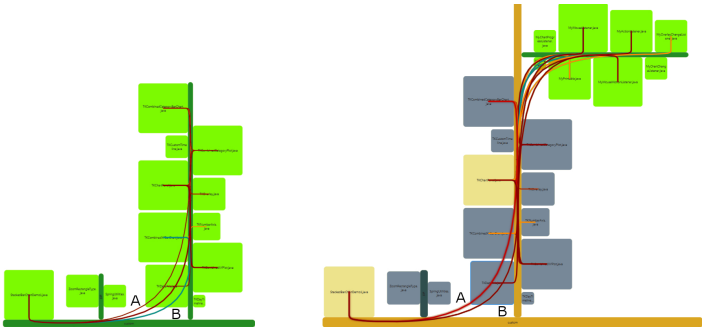
Figure 5.4. Animating particles along an edge

5.2.3 Animating Edge Changes

In the earlier version of our tool [57], we highlighted the evolution of software components through a combination of animated transitions and color interpolations to grow or shrink the city and highlight modification statuses. These transitions take place via the interactive use of a time slider that updates the hierarchical representation of the system to a particular time stamp.

A natural extension while dealing with the exploration of the evolution of relations was to employ a similar mechanism to interpolate the edge representations between time stamps. As such, in our approach nodes that represent software components such as packages and classes may grow, shrink, or change color to depict the insertion, removal, or modification status changes respectively. Similarly, edges may appear, disappear, or be illuminated to show when they are inserted, removed, or modified. The width of the edge is varied to make it appear and disappear, while luminescence is adjusted appropriately to highlight edges that are modified.

This basic concept is depicted in Figure 5.5 where a suburb of the city is examined at two different points in time (*t1* and *t2*). An initial layout is calculated for Figure 5.5a where all nodes are green indicating that they have been inserted into the system. Later, as shown in Figure 5.5b the city grows to indicate the addition of a package and the color of each node depicts its modification status; grey, yellow, and green represents components unchanged, modified, or inserted since the last time stamp. Similarly, the relations of these



(a) Suburb at the first date

(b) Suburb at a later date

Figure 5.5. Animating edge representations

components are also interpolated in between the two time stamps. The edge closest to the *label A* becomes thicker and appears to glow, the edge closest to the *label B* disappears, and several new hierarchical bundles may appear indicating access (orange), call (red), dynamic invocation (yellow), import (cyan), or inheritance (blue) relations to the newly inserted nodes.

5.3 Discussion

The ultimate goal of this work is to assist real-world software architects and developers to interactively gain an overview of changes made to both a software structures' hierarchy and its relationships over time. To achieve this goal we have been working with Product Line Architects at the Fraunhofer IESE to enhance their SAVE tool [192]. SAVE is a state of-the-art tool for analyzing and visualizing software architectures. Its primary feature is architecture compliance checking, i.e., assessing the degree to which an implementation complies to its intended architecture.

We introduce an additional view of the extracted data, in the previous version eCITY, in order to provide an overview of the architectures evolution. A preliminary experiment was conducted where we explored the evolution of the system hierarchy of the Apache Tomcat software [58]. The tasks of this experiment were limited to the

changes made to the hierarchical decomposition of the system over a two-month period. Our approach was well received by the experts who found interactively updating a well-known metaphor such as the software city to be both natural and intuitive. However, we quickly identified the need to examine the architectural relationships of the software system and not just its hierarchical decomposition.

Providing the implementation in our current version of the tool, eCITY+, and applying it on a real system such as the Apache Tomcat (see Figure 5.1) led us to some useful insight as to what works well and what needs to be further improved. In general, applying HEBs on the top of the city was found useful in gaining an overview or trend of how the individual components are connected to one another. However, due to the multitude of edges found in real systems we need to be careful while applying animations to these edges to determine their evolution.

eCITY+ filters out these edges and allows users to interactively monitor their evolution by selecting components. The resulting visualization highlights nodes by interpolating its color and allows users to interactively select a node to show all incoming and outgoing relations. Reserving colors to represent the type of relation limits how eCITY+ represents changes made. An interesting observation was that while interpolating the width to indicate the insertion or removal of an edge shows promise, using luminescence to highlight modifications does not work so well. One idea that we are pursuing is to vary the outline and luminescence in different colors in order to highlight modifications.

Another identified area of improvement in eCITY+ is the efficient handling of multiple edges that may lie on top of each other. Currently, we are experimenting with a combined approach, i.e., on the one hand we may add more control points to some of these edges and relax them for others to create an offset, while on the other we are incorporating interaction techniques to assist users to interactively focus-on and lift each such edge. An example of such might be a combination of the *EdgeLens* [198] and *EdgePlucking* [199] techniques.

5.4 Concluding Remarks

In this chapter we presented the eCITY+ tool that focuses on visualizing the architectural relationships of software systems and not just their hierarchical decomposition.

The goal of this work was to assist architects and developers in examining the evolution of software inter-dependencies for a variety of reasons, such as fixing bugs or to simply get a better impression of the software architecture. As commented by earlier works, meaningful visualizations can be built that represent the hierarchy using an appropriate tree structure and overlay the remaining relations. However, we stress on the need to represent these relations in an uncluttered manner and to examine them interactively.

To meet these requirements we have experimented with an edge bundling technique on top of the popular city metaphor to provide an overview of a software system that depicts both the hierarchical decomposition and the architectural relationships. Further, we have looked into exploiting different animation techniques to highlight various aspects of these relations: particle animation to show the direction of a relation and key frame animation to interactively highlight its evolution.

In informal interviews conducted with experts from visualization, HCI, and software engineering backgrounds we received overall positive feedback about our approach and the usability of our tool. They appreciated the bundling of relations using the streets' space, as it produces a compact visualization with more information. Further, we received many useful recommendations such as providing an interactive magnifying lens facility to assist the users in extracting the required edge; i.e., a means to pick the specified edge and to interact with it individually.

In future work, we aim to address the above recommendations as well as to research how to improve the tracking of relations in a large software system. Using animations is a powerful tool but one that needs to be handled carefully, especially when dealing with a multitude of edge representations. As such, we need to handle our edge representations better and with care to have a universal approach that may be applied to larger systems. We intend to bring our ideas to maturity and conduct a detailed evaluation study in order to judge the efficiency, effectiveness, usability, and acceptability of our solution.

6

Visual Analysis in a Collaborative Environment

" *The secret is to gang up on the problem, rather than each other* "

– Thomas Stallkamp [1]

Most software development tools and applications are designed from a single-user perspective and are bound to the desktop. These tools and applications make it hard for developers to analyze and interact with software artifacts collaboratively. In this chapter, we extend the concept of IVA to a scalable large Hi-Res Tiled-Wall display – thereby, enabling a multi-user multi-application collaborative framework.

In recent times, the relatively low costs for setting up large display systems have led to an highly increased usage of such devices. However, it is equally vital to optimally utilize their size and resolution to effectively explore data through a combination of diverse visualizations and an applicable means of interaction. In this regards, a lightweight dispatcher framework was developed to facilitate input management, focus management, and the execution of several interrelated yet independent visualizations. The approach is deliberately kept flexible to not only tackle different hardware configurations but

[1] Founder and principal of Collaborative Management LLC, a private supply chain consulting firm

also the number of visualization applications to be implemented. This shall be demonstrated further through two different case studies that incorporate interrelated visualizations equally well on both a 3x3 Hi-Res Tiled-Wall as well as a single desktop.

This chapter is organized as follows. Motivation and context are provided in Section 6.1, related work are examined in Section 6.2, the proposed dispatcher framework is described in Section 6.3, and the applicability of our framework is demonstrated in Section 6.4 via two different case studies. Finally, we conclude this chapter in Section 6.5.

6.1 Motivation

The comprehension of software for maintainability purposes is often a social activity and involves developers working within a co-located environment (i.e. in the same room and at the same time). In such scenarios, developers typically work in pairs within a larger team to carry out tasks, such as programming, code reviews, refactoring, and visualization of workflows [200].

Additionally, most software development tools are designed from a single-user perspective and are bound to the desktop. This makes it difficult for users to analyze and interact with software artifacts collaboratively. Further, developers often have to use different applications to explore different software artifacts; for example, in code reviews developers often use different applications to explore test coverage, class dependencies, and class diagrams. It is for these reasons that we propose a multi-user multi-application collaborative framework for use on scalable large Hi-Res displays such as tiled displays.

Innovations in large wall-sized displays have been yielding benefits to visualizations in industry and academia: to cater to a larger audience, for more efficient collaborative work, to further immerse the client in virtual reality applications, and to facilitate visualization of large and complex datasets by maintaining both overview and detail views simultaneously [201].

It is these improvements that have led to the growth of large display implementations despite the limitations in size of a single such display and the costs associated with them. The single most influential factor in this progression has been the advent of tiled display systems - a large display consisting of tiled smaller ones driven by clusters of off-the-shelf PC systems. This leads to complete scalability over the

eventual size of the display while keeping the costs relatively low. These facts have led to a number of research projects that have incorporated tiled displays and made it amongst the top ten visualization research topics in recent times [202].

Most existing tiled-display rendering frameworks distribute geometry and material data to render a single applications graphics database on multiple rendering clients [203, 204], while we aim to facilitate the input management, focus management, and the distributed rendering of several interrelated yet independent applications. The other viable solution would be to have a single scene with various viewports; however, we choose to refrain from this approach to eliminate the possibility of artifacts from one model appearing in the background of another and to a larger extent due to the interaction limitations within these viewports.

In this chapter, we present two different case studies that utilize our framework in order to execute different visualization applications simultaneously, handle interactions between them, and manage input and focus remotely.

6.2 Related Work

For a more comprehensive reading in terms of software engineering collaboration and tools that support this collaboration, the interested reader can inspect the articles of Whitehead [205] and Lanubile et al. [206].

More specifically, in the context of collaborative software engineering, there exist only limited number of studies that have looked at how tools can support collaborative software understanding [207] and collaborative software visualization [200, 208]. However, none have focused on using a CMV approach to handle multi-applications on a tiled wall.

Generally, a collaborative visualization shares control over parameters or products of the scientific or information visualization process [209]. There are a number of application areas where collaborative visualization has played a more dominant role; i.e., multi-player online games, multi-user enabling of single user application, collaborative problem solving tools, and virtual reality environments [208].

In this section, we explore related work with respect to the individual components of our multi-application tiled wall prototype. This includes tiled display rendering frameworks and the distribution and synchronization of input devices and applications over a network. As such, we examine and compare several software solutions that distribute graphics, input devices, and applications.

6.2.1 Distributed Rendering Software

Summarized below are some of the best known distributed rendering software: CAVELib[2], CGLX[3], Chromium [210], Jinx [211], NAVER[4], OpenSG [212], Syzygy [213], VRjuggler [214], and TileRenderer [201]. Additional findings are reported in the surveys [203] and [204].

CAVELib was developed by the Electronic Visualization Lab to run the first Cave [215]. It uses the concept of shared memory to distribute the navigation matrix and the input data through the cluster. According to [216] its major setback is that it uses OpenGL and standard C, hence the requirement for excessive time to code and create a scene. In contrast we utilize scene graphs thus significantly reducing these requirements.

CGLX executes and manages instances of an application on rendering nodes through a thread-based network communication layer. CGLX is currently available only for Linux and Mac Operating Systems, consequently it is not applicable to the widespread Windows platform.

Chromium is an extensive system for interactive scalable rendering that uses clusters of workstations. It accomplishes interactive scalable rendering by intercepting OpenGL API calls and distributing them to the nodes of the cluster. Commercial Chromium-based solutions such as TechViz[5] are available that offer a stable version

[2] CAVELib API (http://www.mechdyne.com/integratedSolutions/software/products/CAVELib/CAVELib.htm)

[3] CGLX: A Cross-Platform Cluster Graphics Library (http://vis.ucsd.edu/mediawiki/index.php/Research_Projects:_CGLX/)

[4] The NAVER framework (http://www.imrc.kist.re.kr/wiki/NAVER_Framework/)

[5] TechViz API (http://www.techviz.net/)

and better performance. Its major disadvantage is that the time needed to sort and split OpenGL commands increases with the number of tiles installed. On the other hand no such synchronization issues were faced while using TileRenderer.

Jinx is a comprehensive distributed virtual-environment browser based on the X3D format. It implements a replication approach to provide support on clusters. To the best of our knowledge X3D has not been widely adopted by proprietary software applications. Conversely, Open Inventor is widely used for a wide range of scientific and engineering visualization systems around the world.

NAVER is a Virtual Reality Theater based on SGI OpenGL Performer. It uses a non-standard scripting language that describes the virtual environment and component nodes responsible for rendering, devices, and applications. NAVER provides libraries for interactions, interfaces, and virtual contents for virtual reality environments. Recently, it has been further extended to support Augmented Reality and Ubiquitous Computing environments. Due to the various libraries needed to build NAVER and the quantity of modules involved, it is an extremely large library making its use highly complex.

OpenSG, in addition to being a scene graph, facilitates the use of multiple asynchronous threads to manipulate scene graphs independently. Further, it provides clustering capabilities by distributing a scene graph to its slaves. However, it has a severe limitation on the file formats it supports: its own OpenSG Binary and Virtual Reality Modeling Language (VRML), even resulting in a program crash if the VRML file size gets too large. Instead, we used Coin3D[6] an open source Open Inventor API and did not face any such restrictions on file size.

Syzygy is a tool for programming Virtual Reality (VR) applications on PC clusters. It has a number of practical elements: filters for encapsulation, sound rendering, virtual devices, Master/Slave framework, etc. Further, it has its own distributed operating system called Phleet. Its major drawback is the high complexity for new application development as opposed to the preferred light-weight toolkit – TileRenderer [201].

[6] Coin3D: 3D Graphics Developer Kit API (http://www.coin3d.org/)

VRjuggler is an Message Passing Interface (MPI)[7] based API for developing VR applications. It employs a client-server data distribution model [217] for sharing graphics data which by design requires high network traffic and as such is not recommended for large data sets. On the other hand our chosen framework utilizes a master-slave model [218] that results in lower network traffic and the ability to work with larger data sets.

TileRenderer is a scalable and light-weight redering library developed to drive arbitrary display configurations and support a variety of stereo modes. It implements a master-slave data distribution mechanism [218], i.e., an instance of the application is executed on each node. Furthermore, it uses a sort-first rendering strategy [219] to assign geometry to the individual tiles of the tiled-display. The main advantage of TileRenderer over other distributed rendering libraries is its high degree of versatility that allows more exotic display configurations. In our work, we have extended TileRenderer to a more generic multi-application framework for tiled-displays. However, we have enhanced its core visualization component to be based on Open Inventor scene graphs instead of OpenGL. This provides us with an improved rendering performance and eliminates the need for primitive drawing commands, state settings, and matrix manipulations.

6.2.2 Distributed Device Data

In the related context of collaborative environments, co-located, and synchronous interactions, developers are often faced with open questions as how to support an application on a shard display with different forms of input devices. In their work, Hilliges et al. [220] derive a number of design guidelines for collaborative systems in interactive environments.

APIs such as CAVELib, Syzygy, and VR Juggler follow similar guidelines to provide embedded support for collaborative interaction between remote applications. In the case of the latter, this is in the form of a module called Gadgeteer that distributes device data across machines and clusters. However, instead of integrating such a heavy-

[7] MPICH2: High-performance and widely portable MPI API (http://www.mcs.anl.gov/research/projects/mpich2/)

weight architecture, we preferred to produce our own streamlined mechanism: switching the *virtual focus* and passing encoded/decoded Simple DirectMedia Layer (SDL)[8] messages is visited later in the chapter.

6.2.3 Distributed Applications

There are a few standards or technologies that are designed towards application integration. Common Object Request Broker Architecture (CORBA) was a revelation in allowing developers to undertake distributed object-oriented programming to integrate diverse applications into heterogeneous distributed systems [221]. Similarly, the High Level Architecture (HLA) [222] has been instrumental in linking disparate simulations. Such frameworks provide rich functionality at the expense of API complexity, so we crafted our own method: each application registering for a message type with the dispatcher and processing it accordingly.

6.3 Methodology

In order to facilitate distributed rendering of applications and inter-application communication, we developed a dispatcher framework. Currently, this framework is configured with TileRenderer, however it can be quite easily ported to any distributed rendering software such as the ones mentioned in Section 6.2.1.

First TileRenderer was modified to work with Coin3D[9], an open source Open Inventor API, instead of pure OpenGL. With the incorporation of scene graphs the environment immediately improved significantly, providing two distinct advantages[10]:

- Improved rendering performance and optimal use of available hardware resources. Scene graphs by nature maintain a "retained" model of the virtual world allowing additional optimizations such as parallel processing culling and drawing, and state sorting

[8] SDL – A cross-platform multimedia library (http://www.libsdl.org/cgi/docwiki.cgi/)

[9] Coin3D: 3D Graphics Developer Kit (http://www.coin3d.org/)

[10] Scenegraphs: Past, present, and future (http://www.realityprime.com/articles/scenegraphs-past-present-and-future)

- Eliminate low-level OpenGL commands such as primitive drawing commands, state settings, and matrix manipulations

The next vital step towards our goal of enhanced Tiled-Wall utilization required the ability to handle multiple applications. This led to the development of the dispatcher framework, whoose main functionality is to act as a centralized message center where all messages are received and forwarded as needed. This is accomplished through the following means:

- Individual applications register for a message type at the *Dispatcher* (see Section 6.3.1).
- *NetMessages* (see Section 6.3.2) are created by the applications, the *MessageHandler* (see Section 6.3.3) connects them to the Dispatcher.
- Appropriate messages are forwarded by the *Dispatcher* only to the subscribers (see Section 6.3.1).
- Input is fed directly to the application running TileRenderer, for all others we incorporate the concept of *Virtual Input* - a special type of message that is an encoded SDL event (see Section 6.3.4).

The final step in setting up the Tile-Wall collaborative environment is to configure the hardware. The basic setup is described in Section 6.3.5, while further details are presented in Sections 6.4.1 and 6.4.2.

Figure 6.1 depicts the logical system structure of the dispatcher framework. The developer may add several disjoint applications to the framework, however these applications need to interface with the common *MessageHandler* component for both interaction and communication purposes. It is through the *MessageHandler* that the different applications are able to communicate with the *Dispatcher*. Here a particular communiqué is worth mentioning; encoded SDL events that are used to pass the *virtual focus*, hence providing the ability to interact with applications that may be running on a remote PC.

A centralized message center that is referred to as the *Dispatcher* is responsible for receiving and forwarding messages if necessary. The communication itself is realized by using sockets that are represented as black and white boxes in Figure 6.1. The distinction here is in the communication direction, a black box stands for an output socket while a white box stands for an input socket.

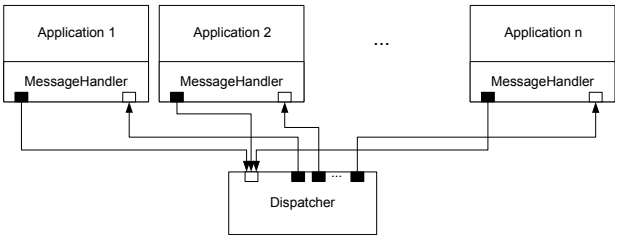


Figure 6.1. Logical structure of Dispatcher Framework

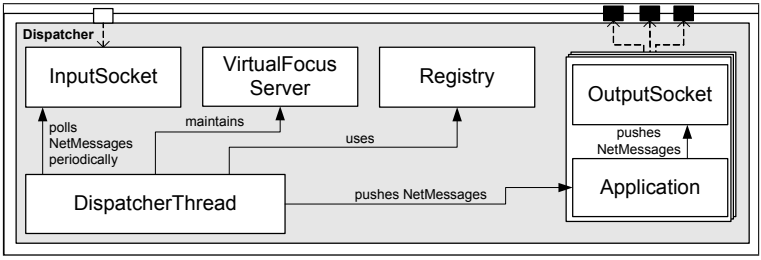
6.3.1 Dispatcher

The main function of the *Dispatcher* to allow applications to register for messages and to forward these messages to the appropriate subscribers. In order to understand this process, we shall examine the logical structure of the *Dispatcher* (see Figure 6.2a) and the flow of its main loop (see Figure 6.2b). The *DispacherThread* is at the heart of the logical structure: it periodically invokes the main loop, polls the *InputSocket*, and maintains the *VirtualFocus* and the *Registry*.

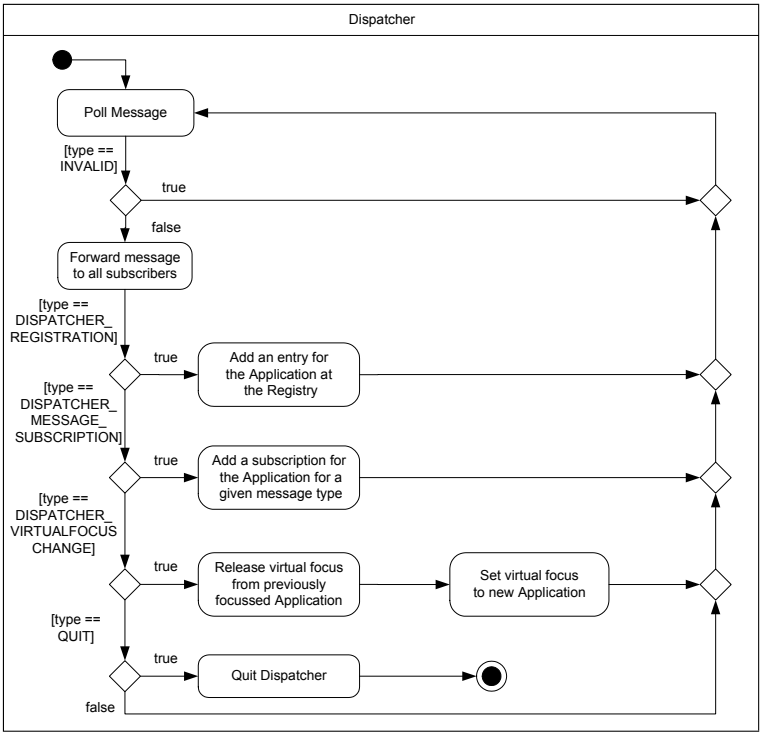
The *DispatcherThread* is an asynchronous thread that implements a non-blocking read by continuously polling the *InputSocket* for an encoded character stream. This encoded stream is used to fill a *NetMessage* data structure, where a key field is the enumerated message type that determines how to process such a message. If the received message is of type *INVALID* this means there is no data available and the next *NetMessage* is polled. This action continues until there is valid data available on the socket - any message that is not of type *INVALID*.

As soon as a valid message is received, it is checked at the *Registry* whether there are subscribers for this type of message. If this is the case, the *NetMessage* is forwarded via the *OutputSocket* of the corresponding registry entry. Depending on the message type, further processing may be required:

DISPATCHER_REGISTRATION If the received message is of this type, a new application has to be registered. This is accomplished through a new application entry created at the *Registry* that contains the *OutputSocket* connection to the remote application.



(a) Logical structure of the Dispatcher



(b) Main loop of the Dispatcher

Figure 6.2. Logical structure and behavior of Dispatcher

DISPATCHER_MESSAGE_SUBSCRIPTION If such a message is received, it implies that an application has subscribed for a new type of message at the *Dispatcher*. Whenever a message with the corresponding type is received by the *Dispatcher*, it will be forwarded to all its subscribers.

DISPATCHER_VIRTUALFOCUSCHANGE This type of message forces the *virtual focus* to switch to another application. The switching process is facilitated by the *VirtualFocusServer* and is responsible for the release of the *virtual focus* from the previous application and for setting it to a new one. In other words, input device data captured by SDL is forwarded to the appropriate subscribing application.

QUIT If a message of type *QUIT* is received, it is distributed to all registered applications in order to initiate their respective shutdown.

If some other type of message is received by the *Dispatcher*, it is simply ignored as it holds no significance for the *Dispatcher* itself.

6.3.2 NetMessage

In order for applications to communicate with one another, they need to pass messages or *NetMessages* to one another. Due to the fact that the *Dispatcher* needs to distinguish between different *NetMessages*, we enumerate these message types. Alternatively, one might have encoded the message type in the text message itself leading to a more complex process. Depending on the type of message, the processing of the message text varies:

INVALID An invalid message can not be processed. Non-blocking socket reading may return NULL, if there is no value to read. To avoid NULL references, this type is introduced.

DISPATCHER_REGISTRATION To take part in the communication process, the applications have to register at the *Dispatcher* by using this type of message.

DISPATCHER_MESSAGE_SUBSCRIPTION To be informed when a certain type of message is received at the *Dispatcher*, a client has to subscribe for it. This is done to avoid passing all messages to all the clients - flooding.

DISPATCHER_VIRTUALFOCUSCHANGE To force a change in which application has the *virtual focus*, this type of message has to be sent.

VIRTUALFOCUS_RECEIVED If an application receives this type of message, it has received the virtual focus. In case of *TileRenderer* this implies processing the *SDL_Events* captured by the input devices locally.

VIRTUALFOCUS_LOST If an application receives this type of message, it has lost the virtual focus. In case of *TileRenderer* this means to forward the *SDL_Events* captured by the input devices to the *Dispatcher*.

SDLEVENT This type of message signals that an *SDL_Event* is encoded in the *NetMessage*.

QUIT An application has to finish its current task and shut down.

There are a number of other application specific messages that can be added to exchange specific information as required by the developer. For example, in the case study presented later in Section 6.4.1, we use the following additional messages:

MCS_SELECTION A minimal cut set was selected.

MCS_BE_PROBABILITY A basic event's probability is sent.

HOLDER_BOUNDS_CHANGED Min and Max bounds change.

HOLDER_VALUES_CHANGED Value of a holder changed.

6.3.3 MessageHandler

It has been shown earlier in Figure 6.1 that all applications use the *MessageHandler* component to communicate with the *Dispatcher*. To understand this process, we shall examine the logical structure of the *MessageHandler* as shown in Figure 6.3a and the typical behavior of an application as in Figure 6.3b.

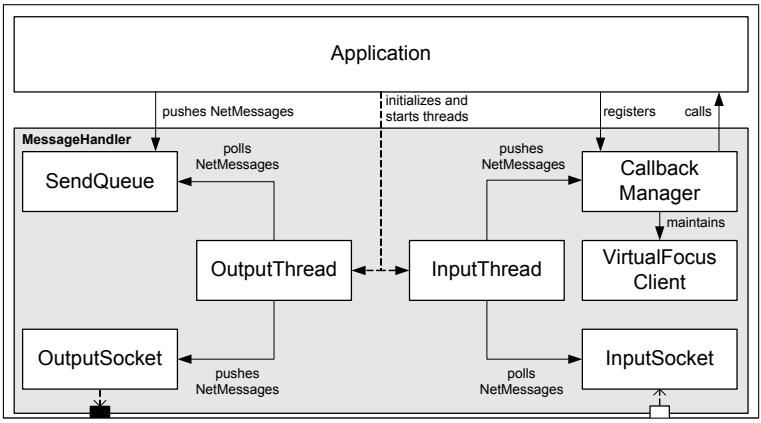
When a new application starts up, it initializes the *MessageHandler*. It is the *MessageHandler* that communicates backwards to the registered applications through the use of callbacks. Thus, the first step is to register all callbacks at the *MessageHandler*, which in turn registers itself automatically for the corresponding message types at the *Dispatcher*. Once these callbacks are registered, both the *InputThread* as well as the *OutputThread* are started.

The original thread continues with the execution of the applications main program loop. While this program loop is in operation, some *NetMessages* may be put into the *SendQueue*. Similarly, the *InputThread* continuously receives messages in a non-blocking way. Much like the functionality described in Section 6.3.1, as long as no valid messages are received the *InputThread* simply waits. On the other hand, a valid message is pushed to the *CallbackManager* as soon as it is received. If there are callbacks registered for this type of message, the corresponding callback function is executed.

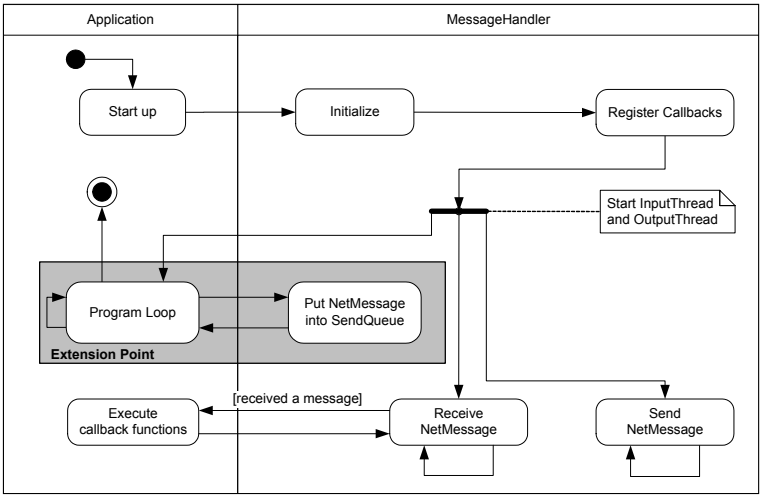
The *VirtualFocusClient* registers callbacks at the *CallbackManager* by default when it starts up. Further, if a received message is of the type *VirtualFocusChange* then the *VirtualFocusClient* is called and the virtual focus is handled automatically - by either grabbing or releasing the virtual focus in question.

The *OutputThread* continuously polls the *SendQueue*. If there are messages available, they are pushed to the *OutputSocket* and transmitted.

Input devices are physically connected only to the *TileRenderer* application, the extension point in Figure 6.3b for this application is refined further in Figure 6.4. The *TileRenderer* based application continuously captures input data from all its connected input devices as *SDL_Events*. If an *SDL_Event* representing a special menu key is pressed, in our implementation the F-1 key, a message is put into the *SendQueue* to switch the focus to the *Menu View*. If the *SDL_Event* represents the special quit key, in this case the ESC key, a message of type *Quit* is put into the *SendQueue*. If the *TileRenderer* holds the



(a) Logical structure of the MessageHandler



(b) Typical behavior of an application during runtime

Figure 6.3. Logical structure and behavior of MessageHandler

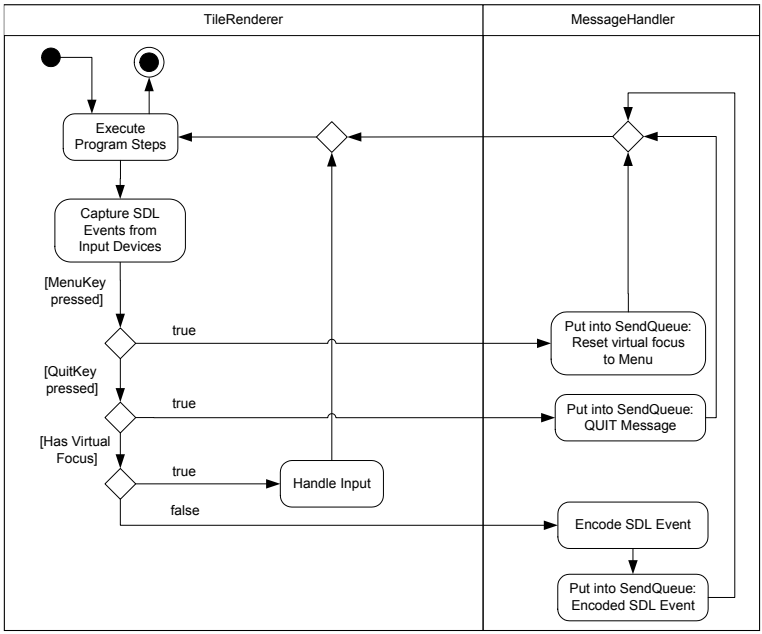


Figure 6.4. TileRenderer and MessageHandler Extention-Point details

virtual focus, it processes these *SDL_Events* locally. Alternatively, if it does not hold the virtual focus then the *SDL_Events* are encoded and put into the *SendQueue*, so that the remote application with the virtual focus can process it.

6.3.4 Virtual Input

The concept of *Virtual Input* was developed to provide input device data to networked machines on a Tiled-Wall. We accomplish this task through the encoding and decoding of SDL events. The structure of an *SDL_Event* is shown in Figure 6.5. This is a streamlined structure as only the *SDL_Event* types which are used by our system are listed,

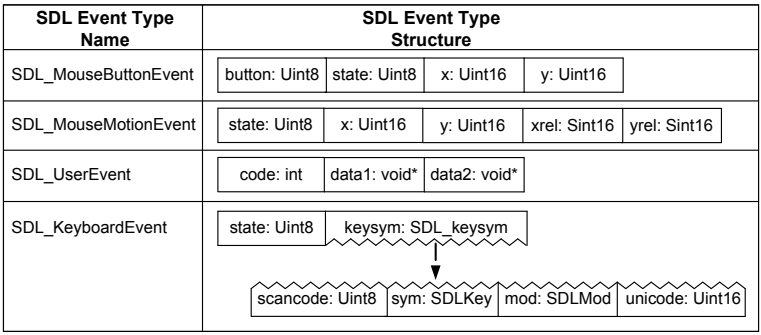


Figure 6.5. Structure of an SDL_Event

further general SDL details are found on their Wiki documentation[11]. It is prominent from this diagram that each *SDL_Event* type has its own specific structure.

SDL_MouseButtonEvent This SDL event type signalizes a mouse button event and is followed by the fields: *button*, *state*, *x*, and *y*. These fields encode the following information: *button* encodes the mouse button index, *state* is either set to *SDL_PRESSED* or *SDL_RELEASED* indicating if the button is pressed or released, *x* holds the x coordinate relative to the window, and *y* holds the y coordinate relative to the window.

SDL_MouseMotionEvent This field signalizes a mouse motion event and is followed by the fields: *state*, *x*, *y*, *xrel*, and *yrel*. These fields encode the following information: for *state*, *x*, and *y* refer above, whereas *xrel* and *yrel* hold the relative motion in the x and y screen directions respectively.

SDL_UserEvent This field refers to a user defined input device and is followed by the fields: *code*, *data1*, and *data2*. These fields encode the following information: *code* is a user defined event code, whereas *data1* and *data2* are user defined data pointers that maybe utilized as required.

[11] SDL: Simple DirectMedia Layer (http://www.libsdl.org/cgi/docwiki.cgi/)

SDL_KeyboardEvent This field refers to a keyboard event and is followed by the fields: *state* and *SDL_keysym*. The *state* field can hold either the value *SDL_PRESSED* or *SDL_RELEASED* indicating if a key is pressed or released. The *SDL_keysym* structure on the other hand is composed of several further fields: *scancode*, *sym*, *mod*, and *unicode*. The *scancode* field is normally not utilized; it contains a hardware-dependent scan-code returned by the keyboard. The *sym* field is the SDL-defined constant that represents the selected key and is often used while programming to inquire if a certain key has been pressed or released. The field *mod* stores the current state of the keyboard modifiers and *unicode* stores the unicode character corresponding to the key if it is enabled.

In the case of *TileRenderer* having virtual focus, these SDL events are processed locally. On the other hand, when another application has the virtual focus we encode these SDL events into a character string and send them to that application via the *Dispatcher*. It is possible to encode and then later decode these SDL events as knowledge of its data structure is known beforehand. Once decoded, a new SDL event is created with the received values and pushed into the SDL event queue of the receiving application, hence the concept of *Virtual Input*. The only exception to the above process was the *SDL_UserEvent*, where *data1* and *data2* are user defined data pointers. An example of this might be adding a space mouse as a user defined device.

6.3.5 Basic Hardware Configuration

The framework presented in this paper is highly flexible in terms of hardware configuration. A text file holds key information for each application such as: the name of the application, the port number associated to it, the PC host name, and a keyword indicating whether it is virtually focus-able. Depending on the desired configuration, the developer would need to carry out the following two tasks for each application:

1. Provide the correct host name in the configuration file
2. Supply appropriate orientation and dimension for each view

In our work we have experimented with a two-monitor single-PC solution as well as a nine-monitor five-PC solution. The hardware employed for these solutions is listed below and more details are presented in the ensuing case studies (see Section 6.4).

Desktop This configuration consisted of a standard monitor with 1920x1200 pixels and a Zalman Trimon passive-stereo monitor with a resolution of 1600x1050. The desktop PC ran on an Windows XP operating system and had the following key components: 2.60 GHz AMD Phenom™ 9950 Quad-Core Processor, 3.25 GB of RAM, and an NVIDIA GeForce GTX 280 graphics card. Additionally, we added the 3Dconnexion Space Navigator as a user defined interaction device.

Tiled-Wall The Tiled-Wall consisted of nine displays controlled by five network-connected PCs. Each computer ran on Windows XP and had the following hardware configuration: Intel® Core™ Dual Core @ 2.4 GHz, 4 GB of RAM, and two GeForce 7950 GX2 graphics cards. Each screen of the Tiled-Wall has a resolution of 2500x1600 pixel, making the total resolution of the wall 7500x4800.

6.4 Case Studies

In this section, we present two different scenarios or case studies where we have implemented our dispatcher framework to support the simultaneous execution of multiple software visualizations. The end result is a seamless CMV-based integration of multiple applications on a large Hi-Res Tiled-Wall display.

These case studies were completely independent of one another and it was equally easy to configure our framework according to the varied needs. On the one hand, we "brushed and linked" applications to assist software engineers in the safety and security of embedded systems. While on the other, we rendered the applications presented in this thesis to support multiple users in a collaborative environment.

6.4.1 Safety and Security Analysis using CakES

The framework presented in Section 6.3 has been used to produce a safety visualization called Cake metaphor for safety analysis of Embedded Systems (CakES) [223] that was designed to assist software engineers in the safety and security of embedded systems. This visualization system consists of multiple applications that visualize the physical model, the minimal cutsets, and the basic events of the fault

tree. Our framework facilitates interaction amongst these different applications, allowing the user to have different levels of focus and context simultaneously. Figure 6.6 shows a real-time screen shot of these applications interacting with one another.

A brief introduction to these safety analysis topics is listed below with references for further readings:

- **Fault Trees (FTs)** are tools in system safety, reliability, and availability studies [224].
- **Basic Events (BEs)** are the lowest-level influence factors in the FT and they are represented as the leaves. The hazard that is examined in the fault tree is called the *top event* which is at its root [225].
- **Minimal Cut Sets (MCSs)** are the unique combinations of BEs that can cause the top event to occur [226].

Multiple Applications

The CakES configuration utilizes four distinct applications, referred to as views henceforth, that are interconnected through our framework. These views are are as follows:

Menu View The menu (see Figure 6.7a) functions as the primary interface between the other applications displayed on the TiledWall. It allows the user to switch the virtual focus between these different views. Additionally, it displays vital statistical data about the selected MCS: size, probability, and details of its BEs. Further, the user may reorganize the MCSs in the *Cake View* according to his preferred criteria by using either the sliders or radio buttons provided.

BEs View The *BEs View* (see Figure 6.7b) is directly related to the *Model* and *Cake* Views. In this view, the hardware components related to the BEs within the selected MCS in the *Cake View* are displayed in more detail. Currently, there are no interaction mechanisms in this view.

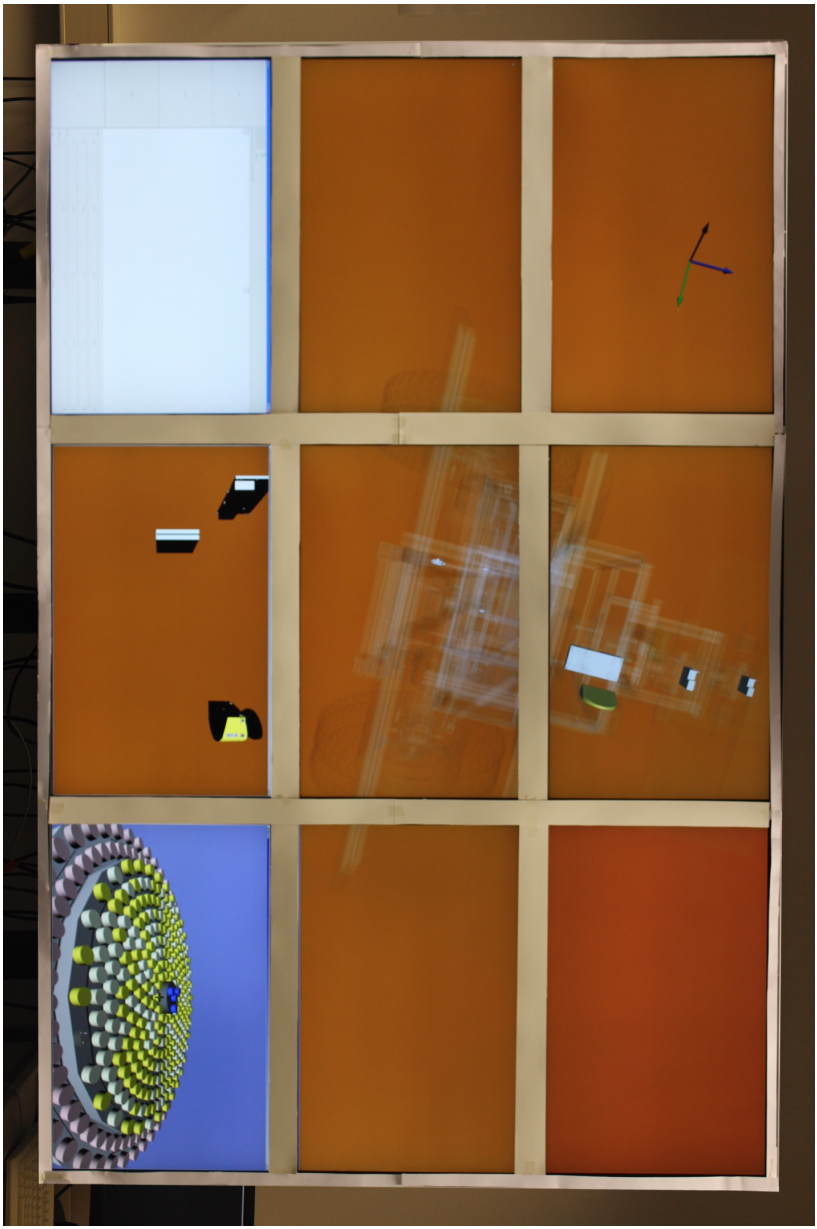
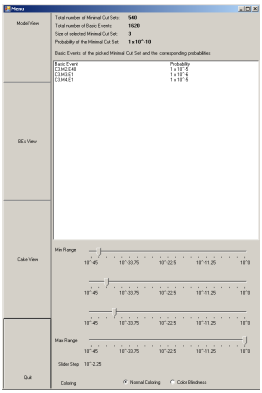
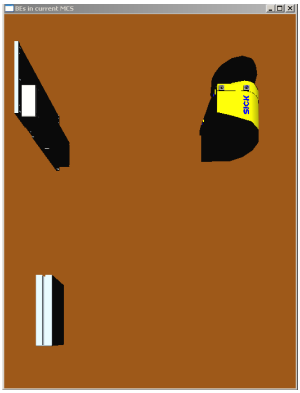


Figure 6.6. Case Study 1: eCITY Tiled-Wall configuration



(a) Menu View



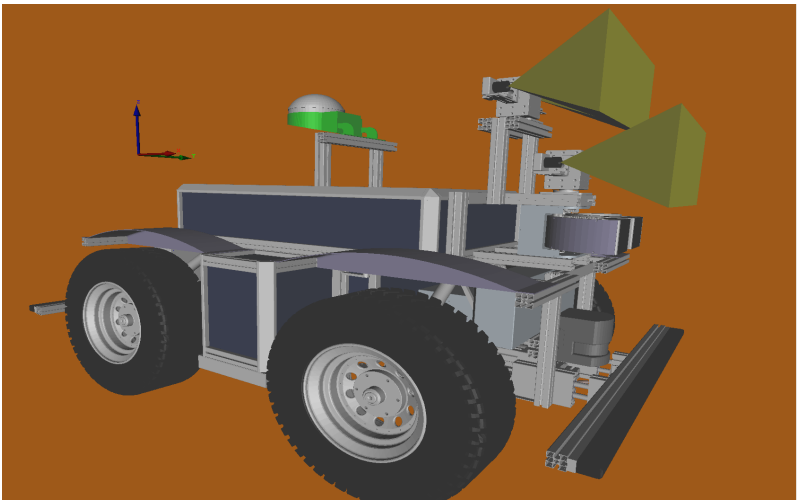
(b) BEs View

Figure 6.7. CakES: Menu and BE Views

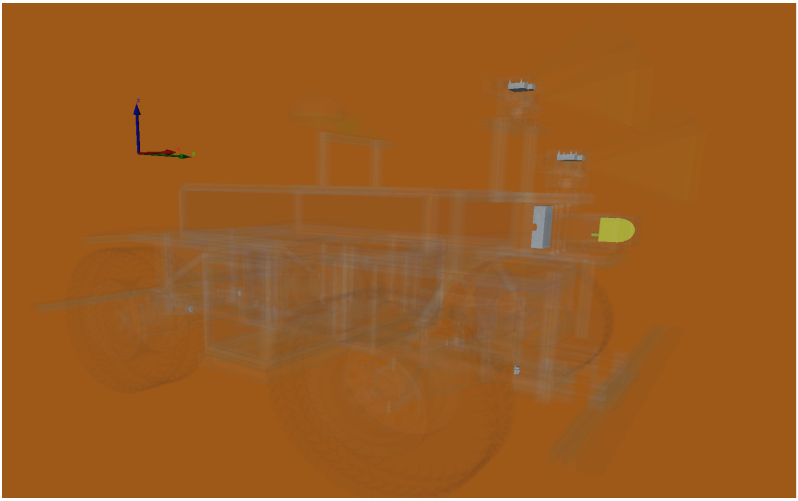
Model View This view is employed to exhibit the physical parts of the Robust Autonomous Vehicle for Off-road Navigation (RAVON) model as shown in Figure 6.8a. The model depicted is of the Robot RAVON, further details can be found on the AG Robotersysteme website[12].

The TileRenderer application drives this view as it is to be displayed both on a stereo monitor as well as on the larger area of a Tiled-Wall. Interaction mechanisms that allow the user to explore the model further are: zooming, rotation, translation, and adding a spotlight. Further, it utilizes the framework by registering for an *MCS_SELECTION*, so that once an MCS is selected in the *Cake View* appropriate data is received. As a consequence of this data exchange, the RAVON model is rendered transparent and the relevant BEs are rendered opaque (see Figure 6.8b).

[12] AG Robotersysteme: Ravon (http://agrosy.informatik.uni-kl.de/en/robots/ravon/)



(a) Before (no MCS selected)



(b) After (MCS selected)

Figure 6.8. CakES: Model View

Cake View This view uses a Cake metaphor [223] to visualize the MCSs, their probabilities, and their BEs. A text file containing information about the BE distribution is generated using the *ESSaRel* tool[13] and used in the initial formation of the Cake.

The Cake consists of three separate levels depicted by red, yellow, and blue cylinders. Each cylinder represents an MCS and within each MCS there are a certain number of BEs. These three levels correspond to a range of fault probabilities that may also be adjusted via the Menu sliders. Further, each level uses saturation to distinguish between probabilities that lie in the same range.

The user is provided interaction mechanisms quite similar to the *Model View*. In addition the concept of *Virtual Picking* was developed to handle virtual interactions within this view, as there was no real focus available - the user interacts through remote devices that send relevant data through our framework. *Virtual Picking* is accomplished by tracking relative mouse movements, drawing a ray through the current mouse position and the far clipping plane, and selecting the intersecting shape. When invoked, it makes an MCS transparent and one can see the BEs within. Information regarding the selected MCS and its BEs are then sent to the other views interested in them through the *Dispatcher* mechanism.

Hardware Configurations

For the CakES visualization, two different configurations were employed. The first is a two-monitor single-PC solution, while the second is a nine-monitor five-PC tiled solution.

Desktop This configuration was built mainly for development purposes and its physical layout is depicted in Figure 6.10a. A stereo monitor was used to display the physical parts of the RAVON model as shown in Figure 6.8. On the other hand, a standard monitor was used to tile the *Menu*, *BEs*, and *Cake* Views next to each other as shown in Figure 6.9.

[13] Background information — ESSaRel (http://www.essarel.de/background/background.html)

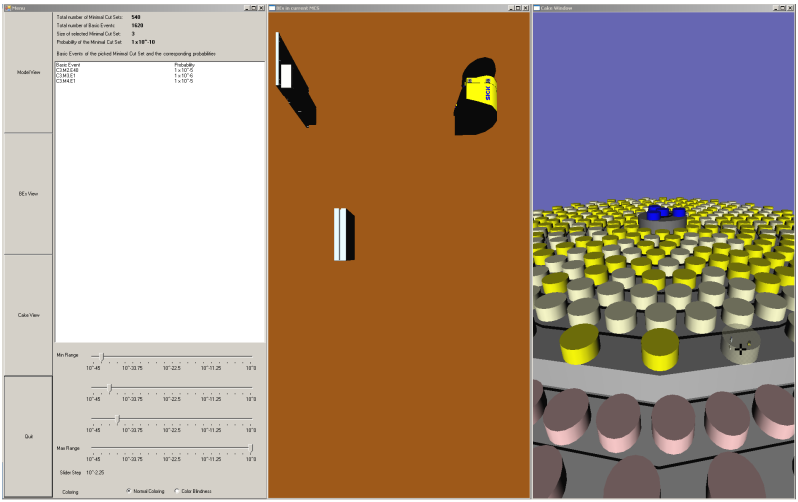
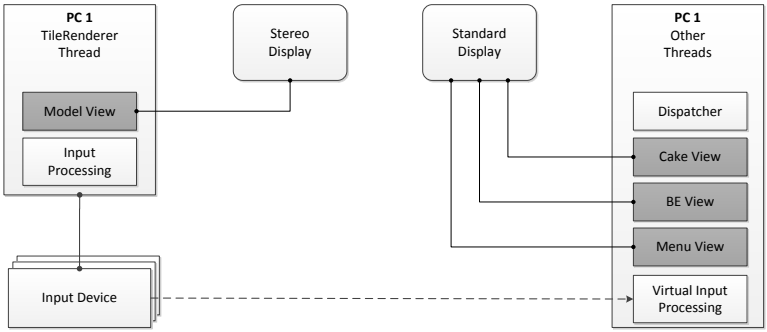


Figure 6.9. CakES: Desktop Views

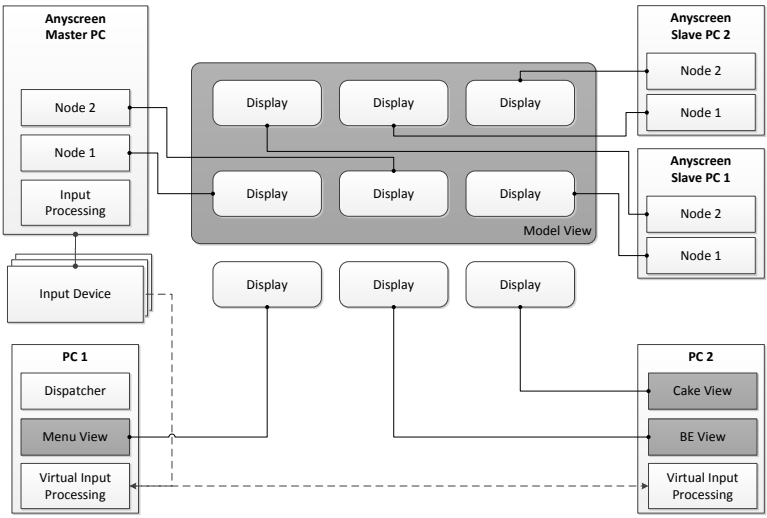
Tiled-Wall A 3x3 Tiled-Wall with five computers was implemented as depicted in Figure 6.10b. This image is based on the 'schematic view on a typical 3x3 tiled display' figure presented by Deller et al. [201], it is modified to incorporate our dispatcher framework. Further, there is an upper and a lower seperation of the Tiled-Wall:

- The upper six displays are managed by *TileRenderer* - it uses three PCs for this task, where one of them acts as the Master-PC and controls the other two Slave-PCs via the network.
- The lower three displays are controlled by the remaining two PCs. One of them runs the *Dispatcher* framework and displays the *Menu View* on the lower-left screen of the Tiled-Wall. The other is used to visualize the *BEs* and *Cake Views* on the remaining two screens.

All the above-mentioned views are controlled by the same input devices, that are physically connected to the Master-PC of the *TileRenderer*. This configuration allows the Master-PC to process input events locally when the *Model View* application has the *virtual*



(a) Config. 1. Desktop



(b) Config. 2. Tiled-Wall

Figure 6.10. CakES: Desktop and Tiled-Wall configurations

focus. On the other hand, it forwards input data via the network to PC1 and PC2 when one of those two applications has the *virtual focus*. A real-time screen shot of the tiled-display was presented earlier in Figure 6.6.

6.4.2 Software Measurement Analysis using VIMETRIK

Earlier, in Chapter 3 we presented our VIMETRIK tool that facilitates users in defining and visualizing software measurements via workflows. Similar to the case study described in Section 6.4.1, this tool consists of multiple applications; namely the KNIME based workflow platform and some custom software visualizations. In this regards, we can leverage our Dispatcher framework to support multiple users in a collaborative environment such as the Tiled-Wall.

In this section, we present work-in-progress where we would like to support multiple co-located users with the above-mentioned applications. The current implementation renders multiple applications on a Tiled-Wall (see Figure 6.11) and handles *Virtual Input*; however, brushing and linking via *NetMessages* is ongoing work. Upon completion of this work, we would like software analysts to work together in creating workflows and examining their findings via the customization of different visualizations. Further, a user study is planned to examine the effectiveness, usability, and acceptability of our approach.

Multiple Applications

Earlier, in Figure 3.7 we depicted a typical scenario where the user models a workflow – we shall refer to this as the **Workflow View**. We also discussed how a user may add a CustomNetworkViewer node to the workflow and generate custom visualizations; the latter are in fact external applications that rely on the workflow generated data.

For the Tiled-Wall version, there is an additional configuration in the CustomNetworkViewer which saves the workflow data on a networked drive. As soon as this data is available on the network-connected PC, the relevant visualization renders it accordingly. One of these visualization possibilities, the **City View** (see Figure 3.9), was discussed earlier in Chapter 3. Additionally, the initial Tiled-Wall configuration consists of a *Menu View* and another visualization or view of the data that we call the *Hyberbolic View*.

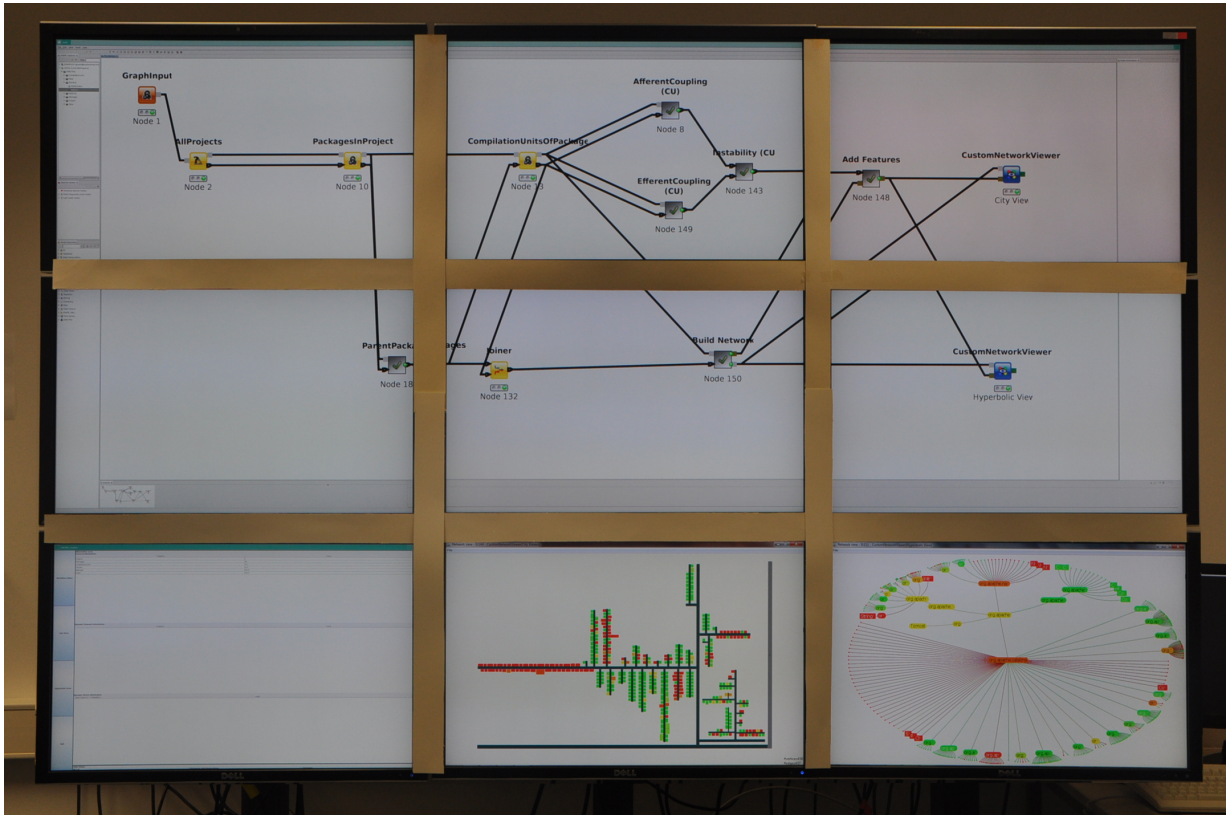
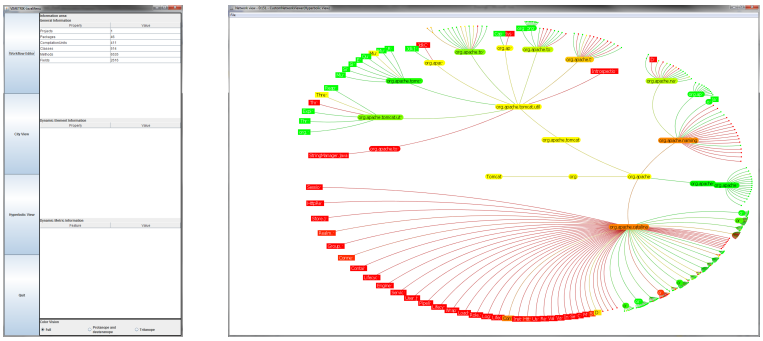


Figure 6.11. Case Study 2: VIMETRIK Tiled-Wall configuration



(a) Menu View (b) Hyperbolic View

Figure 6.12. VIMETRIK: Menu and Hyperbolic Views

Menu View The main function of the menu employed for the VIMETRIK configuration (see Figure 6.12a) is similar to the CakES menu, it is an interface between the applications displayed on the Tiled-Wall. As such, it provides the user a means to switch virtual focus between the different views. In addition, it provides the user with statistical data of the visualization in focus. This data consists of some general information, such as the number of projects, packages, compilation units, classes, methods, and fields; and some interaction based details, such as the relevant properties of the selected entity (see Table 3.1) and the encoded metric values.

Hyperbolic View This view uses a *hyperbolic tree layout* [128] to show the hierarchy of the workflow data. In this example, we feed the visualization data from the workflow of Figure 3.7. More specifically, we provide it with Project->Package, Package->Package, and Package->Compilation Unit hierarchy; and configure it with the instability value[14] as the node color.

Figure 6.12b depicts the distribution of the instability within the Apache Tomcat software system using the Hyperbolic View. A traffic-light metaphor is employed to interpolate values in the [0.0, 0.5, 1.0] range to the colors [green, yellow, red], similar to the

[14] Instability is an indication of the entities resilience to change and is defined by the ratio of efferent to total coupling

City View. This color coding gives an impression of the instability value of each entity; the more the red, the more the instability and the more the green, the more the stability.

In our implementation, varying the node size is avoided as it would adversely affect they layout algorithm. Furthermore, when the graph is deemed too large to be rendered effectively, nodes are pruned together and may be interactively expanded to reveal the sub-tree structure.

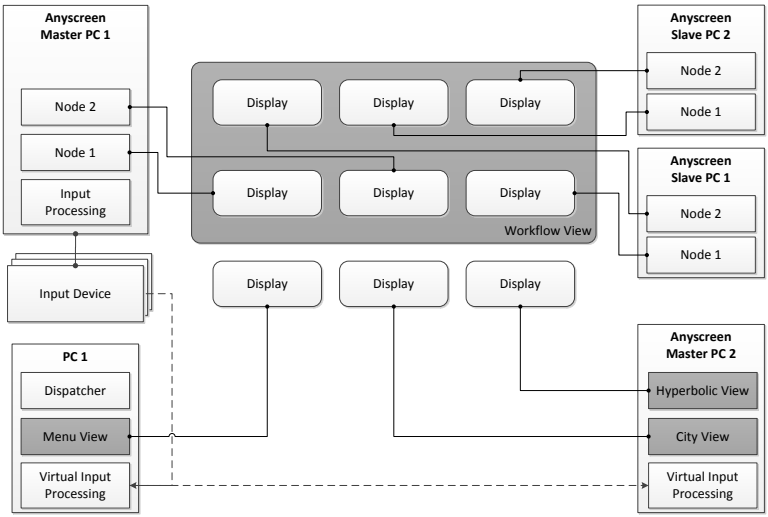
Hardware Configurations

For the VIMETRIK configuration, we have developed two different configurations for the tiled version of the VIMETRIK system. Both are nine-monitor five-PC tiled solutions; the first (see Figure 6.13a) consists of the Menu, Workflow, City, and Hyperbolic Views and the second (see Figure 6.13b) contains the Menu, Workflow, and City Views. The only difference in the two configurations is that the latter uses one more display for the city visualization at the expense of an additional view of the data. As part of future work, we intend to investigate the effect of these two variables on the analysis of a software system.

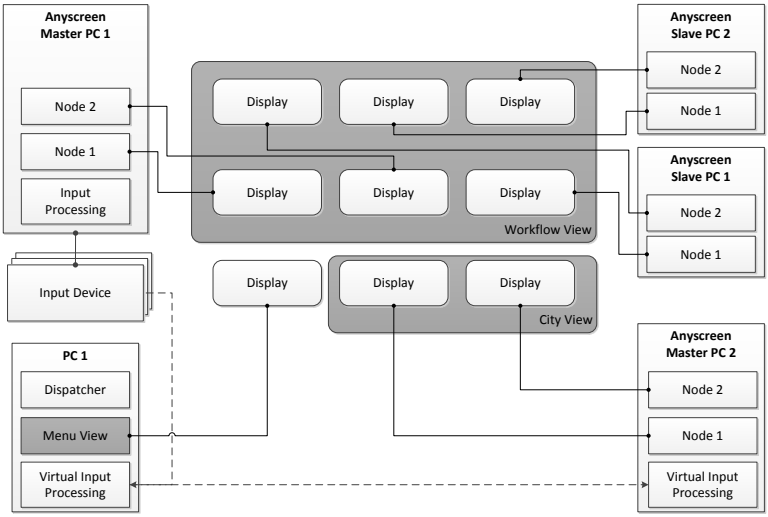
6.5 Concluding Remarks

In this chapter, we presented a light-weight yet flexible multi-application collaborative framework for co-located software development teams to use on a scalable Tiled-Wall display. Our primary design considerations consist of supporting multiple visualizations, distribute the rendering of visualizations across a Tiled-Wall, and support multiple users. These aspects were described in the design of our framework, the various visualizations we have incorporated, and the implementation details of our case studies.

In our work we use *TileRenderer* to handle the tiled display rendering due to its simple yet robust rendering mechanism, however, our framework can be easily ported to other distributed rendering software. As seen in the CakES and VIMETRIK case studies, we were



(a) Config. 1. Workflow, Menu, City, and Hyperbolic Views



(b) Config. 2. Workflow, Menu, and City Views

Figure 6.13. VIMETRIK: Tiled-Wall configurations

able to apply various configurations on a 3x3 Tiled-Wall. The latter is work-in-progress where we are adding brushing and linking techniques to achieve a seamless integration, while the former is part of previous work [227].

In earlier configurations, we used *TileRenderer* to manage the upper six displays and managed three stand-alone visualizations on the lower displays. Recently, as in the case of the alternative VIMETRIK configuration, we have also started looking at distributing the rendering of more than just one application. Ultimately, we would like to study the effect of have more resolution versus an alternative view of the data on collaborative software development tasks.

Further, our framework supports inter-operability between the various applications in order to create a homogeneous CMV solution. To facilitate a seamless integration, applications have to register with our dispatcher and communication with one another via messages. This is a highly adaptive process, where the developer has the complete freedom to adjust the number of applications that may be incorporated; it is simply a matter of adding applications to a configuration file and providing appropriate display parameters. Similar to the work of AlTarawneh et al. [228], we plan to use this mechanism to incorporate smart devices such as smart-phones and tablets to enable multiple users to interact with visual elements on the large display. Thereby, allowing users to freely interact with the visualizations and not be bound to the desktop.

7

Conclusion and Outlook

“ The measure of greatness in a scientific idea is the extent to which it stimulates thought and opens up new lines of research ”

– Paul A.M. Dirac [1]

In this chapter, we summarize the approach and results achieved in this dissertation. Additionally, we elaborate on future research directions in response to questions raised in this work. Finally, we discuss research tasks that benefit from the results of this dissertation.

7.1 Conclusion

In light of the complexity and pervasiveness of software systems nowadays, it is no wonder that understanding software in terms of maintenance and evolution takes up the highest fraction of the total costs of a software system. The importance of understanding the structure and behavior of a software system can especially be seen in performing effective maintenance. Interestingly, modifying a software system implementation without sufficient understanding is likely to

[1] An English theoretical physicist who is regarded as one of the most significant physicists of the 20$^{\text{th}}$ century. Dirac made fundamental contributions to the early development of both quantum mechanics and quantum electrodynamics.

introduce design anomalies, might cause the implementation to degrade, and further complicate any understanding of the system [229]. With the aim of facilitating the comprehension of a software system, a variety of interactive visual analysis concepts have been proposed that have been implemented as industry-ready software analysis tools or integrated into existing software analysis tools. Such tools support software analysts in their tasks of generating software measurements, in their tasks of interactively exploring the resulting multidimensional data, and creating higher-level structural abstractions. At the present time there are hardly any software analysis tools available as commercial products that combine computational analysis and software visualization to provide an understanding of the system's structure and behavior.

In academic research related to software analysis, a variety of visual methods and concepts have been put forward for software understanding and maintenance. However, their prototypical implementations are still far from being adopted by industry [47]. The major obstacle preventing transition from concepts to usable tools is the decoupled nature of software data mining and visualization tools and techniques [14]. Further obstacles deal with: 1) proposed visualization methods and metaphors should not deviate from the current solutions [45], and 2) the ability to monitor, visualize, and interact with large-scale software systems in real-time [230]. The former obstacle stems from visual representations are rarely tailored according to the analysts workflow, while the latter deals with the scalability issue of both computational and cognitive aspects.

This thesis aims to overcome the above-mentioned obstacles through a user-centric approach that combines software data mining and software visualization. Further, in the context of structural evolution we provide the analyst with additional interactive views of his data. The approach and results achieved in these areas are covered in the next sections.

Visual Analysis of Software Measurement Data

Analytics has been effective in the understanding of large volumes of multidimensional data to support better decision making. Organizations commonly apply analytics to business data to describe, predict, and improve business performance. The analysis can be repeated with variable input parameters to represent different boundary conditions

or design choices. Ultimately, the goal of the analysis is to generate useful knowledge from complex multidimensional data. In this regards, IVA is seen as a valuable approach to assist the analyst in this process [231]. Unfortunately, software analysts need better support to complete their daily tasks and make better informed decisions.

The supposition of this dissertation is that by facilitating practitioners with interactive software analytics that summarize trends, synthesize knowledge, and visualize facts and measurements, we can help them make better data-driven development decisions. To validate this claim, we have developed a prototype tool called VIMETRIK. This tool consists of a variety of innovative approaches that have been described in Chapter 3, namely:

- A full extraction of syntactic and semantic details of a software system. In this regards, we present a novel graph database model as well as mechanisms to extract measurement details from the database.
- An interactive visual workflow modeling approach that alleviates the complexities of dealing with the underlying database. As such, the analyst requires no knowledge of the data ontology or the querying mechanisms. Instead, he deals with visual nodes that represent sub-queries and is concerned with their configuration and interconnectivity.
- A plethora of visualization options that consist of traditional views (i.e., tabular views, scatter plots, box plots, histograms, line charts etc.) as well as customizable software visualizations (i.e., city, sunburst, and hyperbolic views). All of these views are configurable and may be attached to any node in the workflow.

The preliminary study we conducted shows both the feasibility and potential of our proposed solution. Graduate students from the related fields of SE, CG, and DBIS who had no knowledge of our data model, querying mechanisms, or software analysis could effectively and intuitively perform basic software analysis tasks, something that would not be possible with current tools. These participants were able to complete software measurement and analysis tasks with an effectiveness (completion and accuracy of tasks) of over 85%. Further, they all found the tool to be highly intuitive (usability and acceptability measures). On average, they gave it a score of 4 out of 5 on a Likert scale.

Visual Analysis of Software Evolution Data

In the context of analyzing software evolution, researchers have devised approaches that consist of two main goals: "inferring causes of problems in a system" and "predicting its future" [51]. However, to be able to analyze a software system, one first needs to create a mental map of it according to the analysis goal [232] (i.e detecting critical software components or predicting the location of future defects). With respect to the analysis of the evolution of software, it is essential to track the structure of the software system to explain and document how a system has evolved to its present state and to predict its future development [46].

There have been some promising solutions proposed in academia with respect to the visualization of software evolution [11, 19, 49, 51, 232], however, not many have not made their way into the mainstream. The supposition of this dissertation is that these approaches have been overlooked as they have not been appropriately tailored to meet the requirements of the analysts. To validate this claim, we have augmented traditional views of evolution-based data in an existing tool with additional interactive views. The resulting tool, eCITY was introduced in Chapter 4 and consists of the following alternative perspectives:

- The Timeline perspective; a managerial view of the evolution data that uses a combination of charts (overview and detail) to highlight changes over time.
- The City perspective; a consistent yet evolving city layout that employs both animated transitions to grow or shrink city suburbs and color interpolations to highlight state changes.

The results of a controlled experiment we conducted shows the significance of employing appropriate software visualization techniques and metaphors in conducting such analysis tasks. In particular, using the eCITY approach participants achieved an average gain of 170% in the efficiency and an average gain of 15% in the effectiveness of basic software architecture evolution tasks. Overall, the results show that our solution was in average as acceptable as the original configuration, but was more efficient, effective, and perceived as more useful.

A direct consequence of this experiment was that the experts identified the need to examine the other inter-dependencies of the software system and not just the evolution of its hierarchical decomposition. In Chapter 5, we presented an extension of our tool called eCITY+ that consists of the following features:

- Overlays these additional inter-dependencies (i.e., access, call, dynamic invocation, import, inheritance, etc.) as HEBs.
- Instead of using traditional means (e.g., arrows or color gradients) to show the directions of these relations, it employs interactive particle animations.
- Similar to its predecessor, it animates both the hierarchical and inter-dependency representations to visually compare changes over a user-defined time period.

Overall, we received positive feedback by experts with CG and SE backgrounds. However, we have also received recommendations to improve the visual tracking of these inter-dependencies with respect to the evolution of large-scale software systems. We have already implemented one of these recommendations that filters out all the HEBs and allows the user to interactively monitor their evolution by selecting components of interest.

Visual Analysis in a Collaborative Environment

In this complementary contribution, we introduced a collaborative framework that expends the principle idea of CMVs to a scalable large Hi-Res Tiled-Wall display. This idea was covered in Chapter 6 and consists of the following components:

- Input management of interaction devices shared across the Tiled-Wall.
- Focus management that facilitates switching between the the tiled views.
- "Brush and link" different views of the data via message passing.

The applicability of our approach is highlighted through two completely independent case studies. On the one hand, we "brushed and linked" applications on the Tiled-Wall to assist software engineers in the safety and security of embedded systems. While on the other hand, we rendered the applications presented in this thesis to support multiple users in a collaborative environment.

Software Visualization in the Industrial Context

In practice, there exists a gap between software visualization research and industrial software engineering. As early as 1998, Jim Foley in the foreword of Stasko's book, "Software Visualization: Programming as a Multimedia Experience" [233], states:

> *My only disappointment with the field is that Software Visualization has not yet had a major impact on the way we teach algorithms and programming or the ways in which we debug programs and systems. While I continue to believe in the promise and potential of SV, it is at the same time the case that SV has not yet had the impact that many have predicted and hoped for.*

Almost two decades later, the situation has not changed much. Only a selected few tools or techniques have found their way into real-world software engineering tools [234–236]. Some software engineers are skeptical on the benefits of Software Visualization where they often ask: "what does a SV tool bring as *measurable* added value to me?" [14], others believe that software visualization has not addressed the right application scenarios [237]. In this regards, there are lessons that can be learned from researchers such as Telea and Voinea who have been working for over 15 years in the field. In their paper "Visual Analysis in Software Maintenance: Challenges and Opportunities" [45], they state:

> *Wider adoption of VA principles in this industry has huge potentials. IT professionals are well aware of the high cost of program understanding [178]. Yet, for increased adoption, software visualization designers should focus more on visualization analysis integration and designing simple visual metaphors that convey precisely and directly the value drivers and way of working of specific user groups. If such aspects are considered, we are convinced that VA will make a significant impact to the software industry.*

In this thesis, we have contributed to the field of software engineering by demonstrating that interactive visual analysis should be better adopted and integrated into software analysis processes as they provide practitioners with powerful means that support more informed data-driven decisions. Similar to the recent work of Endert et al. [238],

we argue for a shift in philosophy from "human in the loop" for visual analytics to a "human is the loop". Our work has been not only about new and innovative ideas, but more importantly, the tailoring and application of these ideas to further the value drivers and ways of working for real-world software analysts. We would like to explicitly stress this aspect through the varied response with respect to the VIMETRIK user interface. While CG experts felt a more sophisticated visualization technique (i.e., tag-clouds [239], visual correlation paradigm [240], etc.) could be employed to access data more quickly, SEs actually appreciated the workflow-based approach as it was more akin to their traditional way of working.

This thesis, while enlightening the central role of such integration, is only a first step towards a broad field of research in interactive software visual analysis.

7.2 Future Work

During the course of this dissertation, we have encountered several promising future research directions. Some of them are ideas on how to overcome limitations of our approach. In the following, we outline possible future directions, discussing when appropriate the emanating shortcomings.

Visual Analysis of Software Measurement Data

In this work, we presented a graph data model capable of capturing full source-code details. This data model has been designed for object-oriented programs like Java and C++, however, software systems nowadays are seldom stand-alone but rather connected to other heterogeneous systems. Further limitations of our graph model include the lack of means to incorporate run-time and rule-based analysis. To address these limitations, we plan to compliment our data model with support for other programming languages, embedded systems, dynamic analysis, and rule-based analysis. In terms of procedural languages such as C, Fortran, and Pascal this would mean different types of nodes (i.e., module instead of class, record instead of object, procedure instead of method, etc.) and relations (i.e., procedure call instead of message). Comprehending embedded systems requires analyses that are based on large amounts of multi-faceted data like units

and maintenance costs, available processor, storage, communication resources in system configurations, adherence to architectural, maintainability, and completeness and extensibility requirements. Similarly, dynamic and rule-based analysis look at different aspects of source code. On the one hand, we plan to develop separate data models to capture syntax and semantic details of procedural languages and embedded systems. While on the other, we plan to extend our data models to handle dynamic and rule-based analysis. In addition to new data models, we plan to develop a meta-data model that combines these concepts for a more complete analysis of software systems.

Another interesting area of future work is to examine a sequence of measurement values from different software versions to monitor trends and change requirements. This could be implemented in the form of specialized visualization nodes that keep track of results and highlight changes. Using this approach the user could simply change the configuration of the *GraphInput* node to an alternative version of the code and re-execute all the nodes in the workflow. Instead of discarding previous results, visualization nodes could append the updated results and highlight changes between versions. An alternative approach would be to augment our data models and workflow nodes to handle delta fact extractions. In this approach, the data models would have to complement facts of the source code with time stamps and workflow nodes would have to consummate time-periods of interest.

Finally and most importantly, we plan to incorporate these ideas and conduct a full-scale comparative study that compares our prototype to existing software analysis tools.

Visual Analysis of Software Evolution Data

We extended an existing software analysis tool with additional views of the architectural data. One of these views was designed to create a mental map of the software architecture and track changes to the hierarchy and its inter-dependency. While we received a positive feedback, there are certain limitations of our approach while dealing with architectures of large-scale software systems. In particular, there are several issues related to the evolution of non-hierarchical software architecture relations. In large software systems, it may become difficult to discern changes due to the large number of relations which

may at times even overlap. To overcome this limitation, we plan to create an offset for relations that have almost identical control points and incorporate *EdgeLens* [198] and *EdgePlucking* [199] interaction techniques to interactively focus-on and lift each such edge.

Another shortcoming was the effectiveness of luminescence to highlight the modification of software architecture inter-dependencies. We plan to incorporate additional measures such as varying the thickness and outline of these relations. Further, once these ideas have been incorporated, we plan to conduct a detailed evaluation study to judge the efficiency, effectiveness, usability, and acceptability of our approach to depict the evolution of architectural inter-dependencies. It would also be interesting to study the effects of adding another co-ordinated view that concentrates only on the relations changed in the selected time period.

Visual Analysis in a Collaborative Environment

Our framework supports the inter-operability between various visualizations to create a homogeneous CMV on a Tiled-Wall. In previous work, we completely integrated the "brushing and linking" of different applications to examine the safety and security of embedded systems. However, this is still work-in-progress with respect to the ideas presented in this paper. Upon completion of this work, we would like software analysts to work together in creating workflows and examining their findings via the customization of different visualizations. Consequently, we plan to evaluate the effectiveness, usability, and acceptability of our approach. Further, it would also be interesting to compare the results to a traditional scenario where software analysts use a single desktop for collaborative purposes.

References

[1] James J. Thomas and Kristin A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Ctr, 2005. ISBN 0769523234. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0769523234.

[2] Yarden Livnat, Theresa-Marie Rhyne, and Matthew H. Samore. Epinome: A visual-analytics workbench for epidemiology data. *IEEE Computer Graphics and Applications*, 32(2):89–95, 2012. URL http://dblp.uni-trier.de/db/journals/cga/cga32.html#LivnatRS12.

[3] Todd Barlow and Padraic Neville. A comparison of 2-d visualizations of hierarchies. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, pages 131–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1342-5. URL http://dl.acm.org/citation.cfm?id=580582.857712.

[4] Slawomir Duszynski, Jens Knodel, and Mikael Lindvall. SAVE: Software Architecture Visualization and Evaluation. In Andreas Winter, Rudolf Ferenc, and Jens Knodel, editors, *CSMR*, pages 323–324. IEEE, 2009. ISBN 978-07695-3589-0. URL http://dblp.uni-trier.de/db/conf/csmr/csmr2009.html#DuszynskiKL09.

[5] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006. ISSN 1077-2626. doi: 10.1109/TVCG.2006.147. URL http://www.computer.org/portal/web/csdl/doi/10.1109/TVCG.2006.147.

[6] Michael Balzer and Oliver Deussen. Level-of-detail visualization of clustered graph layouts. In Seok-Hee Hong and Kwan-Liu Ma, editors, *APVIS*, pages 133–140. IEEE, 2007. ISBN 1-4244-0808-3.

[7] M. Termeer, C. F. J. Lange, A. Telea, and M. R. V. Chaudron. Visual exploration of combined architectural and metric information. In *Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT '05, pages 11–, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9540-9. doi: http://dx.doi.org/10.1109/VISSOF.2005.1684298. URL http://dx.doi.org/10.1109/VISSOF.2005.1684298.

[8] Heorhiy Byelas and Alexandru Telea. Visualizing metrics on areas of interest in software architecture diagrams. In Peter Eades, Thomas Ertl, and Han-Wei Shen, editors, *PacificVis*, pages 33–40. IEEE Computer Society, 2009. ISBN 978-1-4244-4404-5. URL http://dblp.uni-trier.de/db/conf/apvis/pacificvis2009.html#ByelasT09.

[9] Michele Lanza. The evolution matrix: recovering software evolution using software visualization techniques. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, IWPSE '01, pages 37–42, New York, NY, USA, 2001. ACM. ISBN 1-58113-508-4. doi: http://doi.acm.org/10.1145/602461.602467. URL http://doi.acm.org/10.1145/602461.602467.

[10] Guillaume Langelier, Houari Sahraoui, and Pierre Poulin. Exploring the evolution of software quality with animated visualization. In *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, VLHCC '08, pages 13–20, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-1-4244-2528-0. doi: http://dx.doi.org/10.1109/VLHCC.2008.4639052. URL http://dx.doi.org/10.1109/VLHCC.2008.4639052.

[11] Richard Wettel and Michele Lanza. Visual exploration of large-scale system evolution. In Ahmed E. Hassan, Andy Zaidman, and Massimiliano Di Penta, editors, *WCRE*, pages 219–228. IEEE, 2008. URL http://dblp.uni-trier.de/db/conf/wcre/wcre2008.html#WettelL08.

[12] Martin Pinzger, Harald Gall, Michael Fischer, and Michele Lanza. Visualizing multiple evolution metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, SoftVis '05, pages 67–75, New York, NY, USA, 2005. ACM. ISBN 1-59593-073-6. doi: http://doi.acm.org/10.1145/1056018.1056027. URL http://doi.acm.org/10.1145/1056018.1056027.

[13] A. Telea, A. Maccari, and C. Riva. An open toolkit for reverse engineering data visualisation and exploration. In Theo D'Hondt, editor, *Technology of Object-Oriented Languages, Systems and Architectures*, volume 732 of *The Kluwer International Series in Engineering and Computer Science*, pages 76–89. Springer US, 2003. ISBN 978-1-4613-5064-4. doi: 10.1007/978-1-4615-0413-9_6. URL http://dx.doi.org/10.1007/978-1-4615-0413-9_6.

[14] Alexandru Telea and Lucian Voinea. Visual software analytics for the build optimization of large-scale software systems. *Computational Statistics*, 26(4):635–654, 2011. URL http://EconPapers.repec.org/RePEc:spr:compst:v:26:y:2011:i:4:p:635-654.

[15] Lian Wen, Diana Kirk, and R. Geoff Dromey. Software systems as complex networks. In Du Zhang, Yingxu Wang, and Witold Kinsner, editors, *Proceedings of the Six IEEE International Conference on Cognitive Informatics, ICCI 2007, August 6-8, Lake Tahoe, CA, USA*, pages 106–115. IEEE, 2007. ISBN 1-4244-1327-3. doi: 10.1109/COGINF.2007.4341879. URL http://dx.doi.org/10.1109/COGINF.2007.4341879.

[16] Gregory Tassey. The economic impacts of inadequate infrastructure for software testing. Technical Report RTI Project Number 7007.011, National Institute of Standards and Technology, Maryland, USA, 2002. URL http://www.nist.gov/director/planning/upload/report02-3.pdf.

[17] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, 1994. ISBN 0201633612. URL http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1.

[18] Tzilla Elrad, Mehmet Aksit, Gregor Kiczales, Karl Lieberherr, and Harold Ossher. Discussing Aspects of AOP. *Commun. ACM*, 44(10): 33–38, October 2001. ISSN 0001-0782. doi: 10.1145/383845.383854. URL http://doi.acm.org/10.1145/383845.383854.

[19] Mircea Lungu and Michele Lanza. Exploring inter-module relationships in evolving software systems. In René L. Krikhaar, Chris Verhoef, and Giuseppe A. Di Lucca, editors, *CSMR*, pages 91–102. IEEE Computer Society, 2007. ISBN 0-7695-2802-3. URL http://dblp.uni-trier.de/db/conf/csmr/csmr2007.html#LunguL07.

[20] Michael W. Godfrey and Qiang Tu. Evolution in open source software: A case study. In *Proceedings of the International Conference on Software Maintenance (ICSM'00)*, ICSM '00, pages 131–, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0753-0. URL http://dl.acm.org/citation.cfm?id=850948.853411.

[21] T. Klemola and J. Rilling. Modeling comprehension processes in software development. In *Cognitive Informatics, 2002. Proceedings. First IEEE International Conference on*, pages 329–336, August 2002. doi: 10.1109/COGINF.2002.1039314.

[22] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later: Using static analysis to find bugs in the real world. *Commun. ACM*, 53(2):66–75, February 2010. ISSN 0001-0782. doi: 10.1145/1646353.1646374. URL http://doi.acm.org/10.1145/1646353.1646374.

[23] Ulrich Bayer, Andreas Moser, Christopher Kruegel, and Engin Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, 2006. ISSN 1772-9890. doi: 10.1007/s11416-006-0012-2. URL http://dx.doi.org/10.1007/s11416-006-0012-2.

[24] Frank Elberzhager, Jürgen Münch, and Vi Tran Ngoc Nha. A systematic mapping study on the combination of static and dynamic quality assurance techniques. *Inf. Softw. Technol.*, 54(1):1–15, January 2012. ISSN 0950-5849. doi: 10.1016/j.infsof.2011.06.003. URL http://dx.doi.org/10.1016/j.infsof.2011.06.003.

[25] B. McCorkendale, X.F. Tian, S. Gong, X. Zhu, J. Mao, Q. Meng, G.H. Huang, and W.G.E. Hu. Systems and methods for combining static and dynamic code analysis, May 13 2014. URL http://www.google.com/patents/US8726392. US Patent 8,726,392.

[26] G. Larsen, E. Kenneth Hong Fong, David A. Wheeler, and Rama S. Moorthy. State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation. Technical Report IDA Paper P-5061, Institute for Defence Analysis, Virginia, USA, 2014. URL https://www.ida.org/~/media/Corporate/Files/Publications/IDA_Documents/ITSD/2014/P-5061.ashx.

[27] Mario Luca Bernardi and Giuseppe A. Di Lucca. Mining design patterns in object oriented systems by a model-driven approach. In Tai-Hoon Kim, Haeng-Kon Kim, Muhammad Khurram Khan, Kiumi Akingbehin, Wai-Chi Fang, and Dominik Slezak, editors, *Advances in Software Engineering - International Conference (ASEA) 2010, Held as Part of the Future Generation Information Technology Conference (FGIT), 2010, Jeju Island, Korea, December 13-15, 2010. Proceedings*, volume 117 of *Communications in Computer and Information Science*, pages 67–77. Springer, 2010. ISBN 978-3-642-17577-0. doi: 10.1007/978-3-642-17578-7_8. URL http://dx.doi.org/10.1007/978-3-642-17578-7_8.

[28] Istehad Chowdhury and Mohammad Zulkernine. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *J. Syst. Archit.*, 57(3):294–313, March 2011. ISSN 1383-7621. doi: 10.1016/j.sysarc.2010.06.003. URL http://dx.doi.org/10.1016/j.sysarc.2010.06.003.

[29] Ralph Peters and Andy Zaidman. Evaluating the lifespan of code smells using software repository mining. In *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering*, CSMR '12, pages 411–416, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4666-7. doi: 10.1109/CSMR.2012.79. URL http://dx.doi.org/10.1109/CSMR.2012.79.

[30] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.*, 19(2):77–131, March 2007. ISSN 1532-060X. doi: 10.1002/smr.344. URL http://dx.doi.org/10.1002/smr.344.

[31] Quinn Taylor and Christophe Giraud-Carrier. Applications of data mining in software engineering. *Int. J. Data Anal. Tech. Strateg.*, 2 (3):243–257, July 2010. ISSN 1755-8050. doi: 10.1504/IJDATS.2010.034058. URL http://dx.doi.org/10.1504/IJDATS.2010.034058.

[32] Zoltan Konyha. *Interactive Visual Analysis in Automotive Engineering Design*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 1 2013. URL http://www.cg.tuwien.ac.at/research/publications/2013/Konyha_2013_IVA/.

[33] F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, Feb 1973.

[34] Jean-Daniel Fekete, Jarke J. Wijk, John T. Stasko, and Chris North. The value of information visualization. In Andreas Kerren, John T. Stasko, Jean-Daniel Fekete, and Chris North, editors, *Information Visualization*, pages 1–18. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-70955-8. doi: 10.1007/978-3-540-70956-5_1. URL http://dx.doi.org/10.1007/978-3-540-70956-5_1.

[35] Johannes Kehrer. *Interactive Visual Analysis of Multi-faceted Scientific Data*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, March 2011. URL http://www.cg.tuwien.ac.at/research/publications/2011/Kehrer-2011-PhD/.

[36] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. ISBN 1-55860-533-9.

[37] Ben Shneiderman and Benjamin B. Bederson. *The Craft of Information Visualization: Readings and Reflections*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. ISBN 1558609156.

[38] Daniel A. Keim, Florian Mansmann, Jorn Schneidewind, and Hartmut Ziegler. Challenges in visual data analysis. In *Proceedings of the Conference on Information Visualization*, IV '06, pages 9–16, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2602-0. doi: 10.1109/IV.2006.31. URL http://dx.doi.org/10.1109/IV.2006.31.

[39] James J. Thomas and Kristin A. Cook. A visual analytics agenda. *IEEE Comput. Graph. Appl.*, 26(1):10–13, January 2006. ISSN 0272-1716. doi: 10.1109/MCG.2006.5. URL http://dx.doi.org/10.1109/MCG.2006.5.

[40] Daniel A. Keim, Jörn Kohlhammer, Geoffrey Ellis, and Florian Mansmann. *Mastering the Information Age - Solving Problems with Visual Analytics.* Eurographics Association, 2010. ISBN 978-3-905673-77-7.

[41] Michael Wohlfart and Helwig Hauser. Story telling for presentation in volume visualization. In *Proceedings of the 9th Joint Eurographics / IEEE VGTC Conference on Visualization*, EUROVIS'07, pages 91–98, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. ISBN 978-3-905673-45-6. doi: 10.2312/VisSym/EuroVis07/091-098. URL http://dx.doi.org/10.2312/VisSym/EuroVis07/091-098.

[42] L.A. Treinish. Task-specific visualization design. *IEEE Computer Graphics and Applications*, 19:72–77, 1999. ISSN 0272-1716. doi: 10.1109/38.788803. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=788803&isnumber=17095.

[43] M. Petre and E. Quincey. A gentle overview of software visualization. *PPIG News Letter*, pages 1–10, September 2006.

[44] Y. Ghanam and S. Carpendale. A survey paper on software architecture visualization. *Technical Report, University of Calgary*, pages 1–10, June 2008.

[45] Alexandru Telea, Lucian Voinea, and Hans Sassenburg. Visual tools for software architecture understanding: A stakeholder perspective. *IEEE Software*, 27:46–53, 2010. ISSN 0740-7459. doi: http://doi.ieeecomputersociety.org/10.1109/MS.2010.115.

[46] Marco D'Ambros and Michele Lanza. Reverse engineering with logical coupling. In *Proceedings of the 13th Working Conference on Reverse Engineering*, pages 189–198, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2719-1. doi: 10.1109/WCRE.2006.51. URL http://dl.acm.org/citation.cfm?id=1174510.1174729.

[47] Alexandru Telea, Ozan Ersoy, and Lucian Voinea. Visual Analytics in Software Maintenance: Challenges and Opportunities. In Jörn Kohlhammer and Daniel Keim, editors, *EuroVAST 2010: International Symposium on Visual Analytics Science and Tech-*

nology, pages 75–80, Bordeaux, France, 2010. Eurographics Association. ISBN 978-3-905673-74-6. doi: 10.2312/PE/EuroVAST/EuroVAST10/075-080. URL http://diglib.eg.org/EG/DL/PE/EuroVAST/EuroVAST10/075-080.pdf.

[48] Dennie Reniers, Lucian Voinea, and Alexandru Telea. Visual exploration of program structure, dependencies and metrics with solidsx. In *VISSOFT*, pages 1–4. IEEE, 2011. ISBN 978-1-4577-0822-0.

[49] Lucian Voinea and Alexandru Telea. Multiscale and multivariate visualizations of software evolution. In *Proceedings of the 2006 ACM symposium on Software visualization*, SoftVis '06, pages 115–124, New York, NY, USA, 2006. ACM. ISBN 1-59593-464-2. doi: http://doi.acm.org/10.1145/1148493.1148510. URL http://doi.acm.org/10.1145/1148493.1148510.

[50] Marco D'Ambros and Michele Lanza. Visual software evolution reconstruction. *J. Softw. Maint. Evol.*, 21:217–232, May 2009. ISSN 1532-060X. doi: 10.1002/smr.v21:3. URL http://dl.acm.org/citation.cfm?id=1552144.1552146.

[51] Frank Steinbrückner and Claus Lewerentz. Representing development history in software cities. In *Proceedings of the 5th international symposium on Software visualization*, SOFTVIS '10, pages 193–202, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0028-5. doi: http://doi.acm.org/10.1145/1879211.1879239. URL http://doi.acm.org/10.1145/1879211.1879239.

[52] Daniel A. Keim, Florian Mansmann, Jörn Schneidewind, Jim Thomas, and Hartmut Ziegler. Visual analytics: Scope and challenges. In Simeon J. Simoff, Michael H. Böhlen, and Arturas Mazeika, editors, *Visual Data Mining*, pages 76–90. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-71079-0. doi: 10.1007/978-3-540-71080-6_6. URL http://dx.doi.org/10.1007/978-3-540-71080-6_6.

[53] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. Guidelines for using multiple views in information visualization. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 110–119, New York, NY, USA, 2000. ACM. ISBN 1-58113-252-2. doi: 10.1145/345513.345271. URL http://doi.acm.org/10.1145/345513.345271.

[54] Jonathan C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Proceedings of the Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization*, CMV '07, pages 61–71, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2903-8. doi: 10.1109/CMV.2007.20. URL http://dx.doi.org/10.1109/CMV.2007.20.

[55] Taimur Khan, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. Visual Analytics of Software Structure and Metrics. In *Proceddings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT' 2015)*, Bremen, Germany, Sept 2015. To appear.

[56] Taimur Khan, Henning Barthel, Karsten Amrhein, Achim Ebert, and Peter Liggesmeyer. An Interactive Approach for Inspecting Software System Measurements. In *Proceddings of the 15th IFIP TC.13 International Conference on Human-Computer Interaction (INTERACT' 2015)*, Bamberg, Germany, Sept 2015. To appear.

[57] Taimur Khan, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. eCITY: A tool to track software structural changes using an evolving city. In *ICSM*, pages 492–495. IEEE, 2013. URL http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6676473.

[58] Taimur Khan, Henning Barthel, Liliana Guzman, Achim Ebert, and Peter Liggesmeyer. eCITY: Evolutionary Software Architecture Visualization – An Evaluation. In Achim Ebert, Gerrit C. van der Veer, Gitta Domik, Nahum D. Gershon, and Inga Scheler, editors, *Building Bridges: HCI, Visualization, and Non-formal Modeling*, Lecture Notes in Computer Science, pages 201–224. Springer Berlin Heidelberg, 2014. ISBN 978-3-642-54893-2. doi: 10.1007/978-3-642-54894-9_15. URL http://dx.doi.org/10.1007/978-3-642-54894-9_15.

[59] Taimur Khan, Shah Rukh Humayoun, Karsten Amrhein, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. eCITY+: A Tool to Analyze Software Architectural Relations through Interactive Visual Support. In Danny Weyns, editor, *Proceedings of the ECSA 2014 Workshops & Tool Demos Track, European Conference on Software Architecture, 2014, Vienna, Austria*, page 36. ACM, 2014. doi: 10.1145/2642803.2642839. URL http://doi.acm.org/10.1145/2642803.2642839.

[60] Taimur Khan, Shah Rukh Humayoun, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. eCITY+: A Visual Environment for Analysing Software Structure and Evolution. In *Demo Proceedings of AVI 2014, International Working Conference on Advanced Visual Interfaces*. ACM, May 2014.

[61] Taimur Khan, Daniel Schneider, Yasmin Al-Zokari, Dirk Zeckzer, and Hans Hagen. Framework for Comprehensive Size and Resolution Utilization of Arbitrary Displays. In Hans Hagen, editor, *Scientific Visualization: Interactions, Features, Metaphors*, volume 2 of *Dagstuhl Follow-Ups*, pages 144–159. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2011. ISBN 978-3-939897-26-2. doi: http://dx.doi.org/10.4230/DFU.Vol2.SciViz.2011.144. URL http://drops.dagstuhl.de/opus/volltexte/2011/3291.

[62] Daniel A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1): 1–8, January 2002. ISSN 1077-2626. doi: 10.1109/2945.981847. URL http://dx.doi.org/10.1109/2945.981847.

[63] Ben Shneiderman. Inventing discovery tools: Combining information visualization with data mining. *Information Visualization*, 1(1):5–12, March 2002. ISSN 1473-8716. doi: 10.1057/palgrave/ivs/9500006. URL http://dx.doi.org/10.1057/palgrave/ivs/9500006.

[64] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[65] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, VL '96, pages 336–, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7508-X. URL http://dl.acm.org/citation.cfm?id=832277.834354.

[66] Simeon J. Simoff, Michael H. Böhlen, and Arturas Mazeika, editors. *Visual Data Mining - Theory, Techniques and Tools for Visual Analytics*, volume 4404 of *Lecture Notes in Computer Science*. Springer, 2008. ISBN 978-3-540-71079-0.

[67] Maria Cristina Ferreira de Oliveira and Haim Levkowitz. From visual data exploration to visual data mining: A survey. *IEEE Trans. Vis. Comput. Graph.*, 9(3):378–394, 2003. URL http://dblp.uni-trier.de/db/journals/tvcg/tvcg9.html#OliveiraL03.

[68] Stephan Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. ISBN 3540465049.

[69] Pak Chung Wong and Jim Thomas. Visual analytics. *IEEE Computer Graphics and Applications*, 24(5):20–21, 2004. doi: 10.1109/MCG.2004.39. URL http://doi.ieeecomputersociety.org/10.1109/MCG.2004.39.

[70] Daniel J. Quinlan. Rose: Compiler support for object-oriented frameworks. *Parallel Processing Letters*, 10(2/3):215–226, 2000.

[71] Rudolf Ferenc, Árpád Beszédes, Mikko Tarkiainen, and Tibor Gyimóthy. Columbus - reverse engineering tool and schema for c++. In *ICSM*, pages 172–181. IEEE Computer Society, 2002. ISBN 0-7695-1819-2.

[72] Warren Harrison. A flexible method for maintaining software metrics data: a universal metrics repository. *Journal of Systems and Software*, 72(2):225 – 234, 2004. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/S0164-1212(03)00092-X. URL http://www.sciencedirect.com/science/article/pii/S016412120300092X.

[73] Jaroslav Pokorny. Nosql databases: A step to database scalability in web environment. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, iiWAS '11, pages 278–283, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0784-0. doi: 10.1145/2095536.2095583. URL http://doi.acm.org/10.1145/2095536.2095583.

[74] Ameya Nayak, Anil Poriya, and Dikshay Poojary. Article: Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19, March 2013. Published by Foundation of Computer Science, New York, USA.

[75] Jürgen Ebert and Daniel Bildhauer. Reverse engineering using graph queries. In Gregor Engels, Claus Lewerentz, Wilhelm Schäfer, Andy Schürr, and Bernhard Westfechtel, editors, *Graph Transformations and Model-Driven Engineering*, volume 5765 of *Lecture Notes in Computer Science*, pages 335–362. Springer, 2010. ISBN 978-3-642-17321-9. doi: 10.1007/978-3-642-17322-6. URL http://dblp.uni-trier.de/db/conf/birthday/nagl2010.html#EbertB10.

[76] V. Winter, C. Reinke, and J. Guerrero. Sextant: A tool to specify and visualize software metrics for java source-code. In *Emerging Trends in Software Metrics (WETSoM), 2013 4th International Workshop on*, pages 49–55, May 2013. doi: 10.1109/WETSoM.2013.6619336.

[77] Norman Murray, Norman Paton, and Carole Goble. Kaleidoquery: a visual query language for object databases. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '98, pages 247–257, New York, NY, USA, 1998. ACM. doi: 10.1145/948496.948529. URL http://doi.acm.org/10.1145/948496.948529.

[78] Mike Cammarano, Xin Luna Dong, Bryan Chan, Jeff Klingner, Justin Talbot, Alon Y. Halevy, and Pat Hanrahan. Visualization of heterogeneous data. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1200–1207, 2007.

[79] Gary Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, April 2006. ISSN 0001-0782. doi: 10.1145/1121949.1121979. URL http://doi.acm.org/10.1145/1121949.1121979.

[80] Y. Livnat, J. Agutter, S. Moon, R.F. Erbacher, and S. Foresti. A visualization paradigm for network intrusion detection. In *IEEE Workshop on Information Assurance and Security 2005*, pages 30–37, June 2005. URL http://www.sci.utah.edu/publications/yarden05/VisAlert.pdf.

[81] Geoffrey Draper, Yardin Livnat, and Richard Riesenfeld. A visual query language for correlation discovery and management. In *Proceedings of the 2nd Annual Visual and Iconic Language Conference*, VaIL '08, 2008.

[82] Per H. Gesteland, Yarden Livnat, Nathan Galli, Matthew H. Samore, and Adi V. Gundlapalli. The epicanvas infectious disease weather map: an interactive visual exploration of temporal and spatial correlations. *JAMIA*, 19(6):954–959, 2012. URL http://dblp.uni-trier.de/db/journals/jamia/jamia19.html#GestelandLGSG12.

[83] Ferdinando Villa, Matthew A. Wilson, Rudolf de Groot, Steven Farber, Robert Costanza, and Roelof M.J. Boumans. Designing an integrated knowledge base to support ecosystem services valuation. *Ecological Economics*, 41(3):445 – 456, 2002. ISSN 0921-8009. doi: 10.1016/S0921-8009(02)00093-9. URL http://www.sciencedirect.com/science/article/pii/S0921800902000939.

[84] Steve Jones. Graphical query specification and dynamic result previews for a digital library. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*, UIST '98, pages 143–151, New York, NY, USA, 1998. ACM. ISBN 1-58113-034-1. doi: 10.1145/288392.288595. URL http://doi.acm.org/10.1145/288392.288595.

[85] D. Young and B. Shneiderman. A graphical filter/flow model for boolean queries: an implementation and experiment. *Journal of The American Society for Information Science and Technology*, 1993.

[86] Niklas Elmqvist, John Stasko, and Philippas Tsigas. Datameadow: a visual canvas for analysis of large-scale multivariate data. *Information Visualization*, 7(1):18–33, March 2008. ISSN 1473-8716. doi: 10.1145/1391107.1391110. URL http://dx.doi.org/10.1145/1391107.1391110.

[87] Anselm Spoerri. Infocrystal: A visual tool for information retrieval & management. In Bharat K. Bhargava, Timothy W. Finin, and Yelena Yesha, editors, *CIKM*, pages 11–20. ACM, 1993. ISBN 0-89791-626-3.

[88] Jiwen Huo. Kmvql: a visual query interface based on karnaugh map. In Stefano Levialdi, editor, *AVI*, pages 243–250. ACM Press, 2008. ISBN 978-1-60558-141-5.

[89] Stavros Polyviou, George Samaras, and Paraskevas Evripidou. A relationally complete visual query language for heterogeneous data sources and pervasive querying. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 471–482, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2285-8. doi: 10.1109/ICDE.2005.12. URL http://dx.doi.org/10.1109/ICDE.2005.12.

[90] Vineet Sinha and David R. Karger. Magnet: Supporting navigation in semistructured data environments. In Fatma Özcan, editor, *SIGMOD Conference*, pages 97–106. ACM, 2005. ISBN 1-59593-060-4.

[91] Agathoniki Trigoni. Interactive query formulation in semistructured databases. In *Proceedings of the 5th International Conference on Flexible Query Answering Systems*, FQAS '02, pages 356–369, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-00074-7. URL http://dl.acm.org/citation.cfm?id=645424.652301.

[92] Peter Lucas, Cristina C. Gomberg, and Steven F. Roth. Visage: Dynamic information exploration. In Michael J. Tauber, editor, *CHI Conference Companion*, pages 19–20. ACM, 1996. ISBN 0-89791-832-0.

[93] Matthew O. Ward. Xmdvtool: integrating multiple methods for visualizing multivariate data. In *Proceedings of the conference on Visualization '94*, VIS '94, pages 326–333, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. ISBN 0-7803-2521-4. URL http://dl.acm.org/citation.cfm?id=951087.951146.

[94] Michael Stonebraker. Visionary: A next generation visualization system for databases. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *SIGMOD Conference*, page 635. ACM, 2003. ISBN 1-58113-634-X.

[95] Maria Cristina Ferreira de Oliveira and Haim Levkowitz. From visual data exploration to visual data mining: A survey. *IEEE Trans. Vis. Comput. Graph.*, 9(3):378–394, 2003.

[96] Chris Stolte, Diane Tang, and Pat Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional databases. *Commun. ACM*, 51(11):75–84, 2008.

[97] Benedikt Stehno. Rapid visualization development based on visual programming. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 2011. URL http://www.cg.tuwien.ac.at/research/publications/2011/stehno-2011-RVD/.

[98] J. Paul Morrison. *Flow-Based Programming, 2nd Edition: A New Approach to Application Development*. CreateSpace, Paramount, NY, USA, 2010. ISBN 978-1451542325.

[99] Axel Wickenkamp Jürgen Münch. M-System NT – Ein flexibles, datenbank-basiertes Mess- und Analyse-System. In *Proceedings of the DASMA Software Metric Congress (MetriKon 2005): Magdeburger Schriften zum Empirischen Software Engineering*, pages 55–64, Kaiserslautern, Germany, November 14-16 2005. Shaker Verlag. ISBN 3-8322-4615-0.

[100] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. In Eytan Adar, Matthew Hurst, Tim Finin, Natalie S. Glance, Nicolas Nicolov, and Belle L. Tseng, editors, *ICWSM*. The AAAI Press, 2009. ISBN 978-1-57735-421-5. URL http://dblp.uni-trier.de/db/conf/icwsm/icwsm2009.html#BastianHJ09.

[101] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exper.*, 30(11):1203–1233, September 2000. ISSN 0038-0644. doi: 10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.3.CO;2-E. URL http://dx.doi.org/10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.3.CO;2-E.

[102] Christopher Eric Weaver. *Improvise: A User Interface for Interactive Construction of Highly-coordinated Visualizations*. PhD thesis, University of Wisconsin at Madison, Madison, WI, USA, 2006.

[103] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, November 2009. ISSN 1077-2626. doi: 10.1109/TVCG.2009.174. URL http://dx.doi.org/10.1109/TVCG.2009.174.

[104] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL http://www.R-project.org. ISBN 3-900051-07-0.

[105] Jean-Daniel Fekete. The infovis toolkit. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '04, pages 167–174, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8779-3. doi: 10.1109/INFOVIS.2004.64. URL http://dx.doi.org/10.1109/INFOVIS.2004.64.

[106] John R. Harger and Patricia J. Crossno. Comparison of open-source visual analytics toolkits. *Proc. SPIE*, 8294:82940E–82940E–10, 2012. doi: 10.1117/12.911901. URL http://dx.doi.org/10.1117/12.911901.

[107] Leishi Zhang, Andreas Stoffel, Michael Behrisch, Sebastian Mittelstädt, Tobias Schreck, René Pompl, Stefan Weber, Holger Last, and Daniel A. Keim. Visual analytics for the big data era - a comparative review of state-of-the-art commercial systems. In *IEEE VAST*, pages 173–182. IEEE Computer Society, 2012. ISBN 978-1-4673-4752-5. URL http://dblp.uni-trier.de/db/conf/ieeevast/ieeevast2012.html#ZhangSBMSPWLK12.

[108] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel, and Bernd Wiswedel. KNIME - the Konstanz Information Miner: Version 2.0 and Beyond. *SIGKDD Explor. Newsl.*, 11(1):26–31, November 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656280. URL http://doi.acm.org/10.1145/1656274.1656280.

[109] K Gallagher, A Hatch, and M Munro. Software architecture visualization : an evaluation framework and its application. *IEEE Transactions on Software Engineering*, 34(2):260–270, 2008. URL http://dx.doi.org/10.1109/TSE.2007.70757.

[110] A. Hatch. *Software Architecture Visualization.* Phd dissertation, University of Durham, 2004.

[111] Pierre Caserta and Olivier Zendra. Visualization of the static aspects of software: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 17:913–933, 2011. ISSN 1077-2626. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.110.

[112] Andreas Kerren, Achim Ebert, and Jörg Meyer, editors. *Human-Centered Visualization Environments.* Lecture Notes in Computer Science, LNCS. Springer-Verlag Gmbh, 1 edition, 2007. ISBN 978-3540719489.

[113] Daniel Archambault, Tamara Munzner, and David Auber. Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, 14:900–913, 2008. ISSN 1077-2626. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2008.34.

[114] Sazzadul Alam and Philippe Dugerdil. Evospaces: 3d visualization of software architecture. In *SEKE*, pages 500–505. Knowledge Systems Institute Graduate School, 2007. ISBN 1-891706-20-9.

[115] Michael Balzer and Oliver Deussen. Hierarchy based 3d visualization of large software structures. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 598.4–, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8788-0. doi: http://dx.doi.org/10.1109/VISUAL.2004.39. URL http://dx.doi.org/10.1109/VISUAL.2004.39.

[116] Andrian Marcus, Louis Feng, and Jonathan I. Maletic. Comprehension of software analysis data using 3d visualization. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, IWPC '03, pages 105–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1883-4. URL http://dl.acm.org/citation.cfm?id=851042.857046.

[117] Jun Rekimoto and Mark Green. The information cube: Using transparency in 3d information visualization. In *In Proceedings of the Third Annual Workshop on Information Technologies & Systems (WITS'93*, pages 125–132, 1993.

[118] J.D. Mackinlay. Opportunities for information visualization. *Computer Graphics and Applications, IEEE*, 20(1):22 –23, jan/feb 2000. ISSN 0272-1716. doi: 10.1109/38.814540.

[119] Raimund Dachselt and Jürgen Ebert. Collapsible cylindrical trees: A fast hierarchical navigation technique. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, pages 79–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1342-5. URL http://dl.acm.org/citation.cfm?id=580582.857708.

[120] Brian Johnson and Ben Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press. ISBN 0-8186-2245-8. URL http://dl.acm.org/citation.cfm?id=949607.949654.

[121] Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, SoftVis '05, pages 165–172, New York, NY, USA, 2005. ACM. ISBN 1-59593-073-6. doi: http://doi.acm.org/10.1145/1056018.1056041. URL http://doi.acm.org/10.1145/1056018.1056041.

[122] Stéphane Ducasse, Simon Denier, Françoise Balmas, Alexandre Bergel, Jannik Laval, Karine Mordal-Manet, and Fabrice Bellingard. Visualization of Practices and Metrics (Workpackage 1.2). Research report, Squale Consortium, March 2010. URL http://hal.inria.fr/inria-00533618/en/.

[123] Weixin Wang, Hui Wang, Guozhong Dai, and Hongan Wang. Visualization of large hierarchical data by circle packing. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 517–520, New York, NY, USA, 2006. ACM. ISBN 1-59593-372-7. doi: http://doi.acm.org/10.1145/1124772.1124851. URL http://doi.acm.org/10.1145/1124772.1124851.

[124] Hans-Jorg Schulz, Steffen Hadlak, and Heidrun Schumann. The design space of implicit hierarchy visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 17:393–411, April 2011. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2010.79. URL http://dx.doi.org/10.1109/TVCG.2010.79.

[125] John Stasko and Eugene Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proceedings of the IEEE Symposium on Information Vizualization 2000*, INFOVIS '00, pages 57–, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0804-9. URL http://dl.acm.org/citation.cfm?id=857190.857683.

[126] Keith Andrews and Helmut Heidegger. Information slices : Visualising and exploring large hierarchies using cascading , semi-circular discs. *Information Visualization*, pages 9–12, 1998.

[127] John Stasko. An evaluation of space-filling information visualizations for depicting hierarchical structures. *Int. J. Hum.-Comput. Stud.*, 53: 663–694, November 2000. ISSN 1071-5819. doi: 10.1006/ijhc.2000.0420. URL http://dl.acm.org/citation.cfm?id=362757.365438.

[128] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-84705-1. doi: http://dx.doi.org/10.1145/223904.223956. URL http://dx.doi.org/10.1145/223904.223956.

[129] Hans-Jorg Schulz and Heidrun Schumann. Visualizing graphs - a generalized view. In *Proceedings of the conference on Information Visualization*, pages 166–173, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2602-0. doi: 10.1109/IV.2006.130. URL http://dl.acm.org/citation.cfm?id=1153927.1154662.

[130] Dirk Zeckzer. Visualizing software entities using a matrix layout. In *Proceedings of the 5th international symposium on Software visualization*, SOFTVIS '10, pages 207–208, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0028-5. doi: http://doi.acm.org/10.1145/1879211.1879243. URL http://doi.acm.org/10.1145/1879211.1879243.

[131] Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. Using dependency models to manage complex software architecture. In *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '05, pages 167–176, New York, NY, USA, 2005. ACM. ISBN 1-59593-031-0. doi: http://doi.acm.org/10.1145/1094811.1094824. URL http://doi.acm.org/10.1145/1094811.1094824.

[132] Holger Eichelberger. *Aesthetics and automatic layout of UML class diagrams*. PhD thesis, Universität Würzburg, Am Hubland, 97074 Würzburg, 2005.

[133] Helen C. Purchase, Jo-Anne Allder, and David A. Carrington. User preference of graph layout aesthetics: A uml study. In *Proceedings of the 8th International Symposium on Graph Drawing*, GD '00, pages 5–18, London, UK, 2001. Springer-Verlag. ISBN 3-540-41554-8. URL http://dl.acm.org/citation.cfm?id=647552.729416.

[134] Dabo Sun and Kenny Wong. On evaluating the layout of uml class diagrams for program comprehension. In *Proceedings of the 13th International Workshop on Program Comprehension*, pages 317–326, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2254-8. doi: 10.1109/WPC.2005.26. URL http://dl.acm.org/citation.cfm?id=1058432.1059369.

[135] Martin Pinzger, Katja Graefenhain, Patrick Knab, and Harald C. Gall. A tool for visual understanding of source code dependencies. In *Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*, ICPC '08, pages 254–259, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3176-2. doi: http://dx.doi.org/10.1109/ICPC.2008.23. URL http://dx.doi.org/10.1109/ICPC.2008.23.

[136] Jean-Daniel Fekete, David Wang, Niem Dang, Aleks Aris, and Catherine Plaisant. Overlaying graph links on treemaps. *IEEE Symposium on Information Visualization Conference Compendium (demonstration)*, 5, 2003.

[137] Alexander Fronk, Armin Bruckhoff, and Michael Kern. 3d visualisation of code structures in java software systems. In *Proceedings of the 2006 ACM symposium on Software visualization*, SoftVis '06, pages 145–146, New York, NY, USA, 2006. ACM. ISBN 1-59593-464-2. doi: http://doi.acm.org/10.1145/1148493.1148515. URL http://doi.acm.org/10.1145/1148493.1148515.

[138] C. Russo dos Santos, P. Gros, P. Abel, D. Loisel, N. Trichaud, and J. P. Paris. Metaphor-aware 3d navigation. In *Proceedings of the IEEE Symposium on Information Vizualization 2000*, INFOVIS '00, pages 155–, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0804-9. URL http://dl.acm.org/citation.cfm?id=857190.857697.

[139] Thomas Panas, Rebecca Berrigan, and John Grundy. A 3d metaphor for software production visualization. *Proceedings on Seventh International Conference on Information Visualization 2003 IV 2003*, 314: 314–319, 2003. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1217996.

[140] Thomas Panas, Thomas Epperly, Daniel Quinlan, Andreas Saebjornsen, and Richard Vuduc. Communicating software architecture using a unified single-view visualization. In *Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems*, pages 217–228, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2895-3. doi: 10.1109/ICECCS.2007.20. URL http://dl.acm.org/citation.cfm?id=1270390.1271065.

[141] Hamish Graham, Hong Yul Yang, and Rebecca Berrigan. A solar system metaphor for 3d visualisation of object oriented software metrics. In *Proceedings of the 2004 Australasian symposium on Information Visualisation - Volume 35*, APVis '04, pages 53–59, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc. ISBN 1-920682-17-1. URL http://dl.acm.org/citation.cfm?id=1082101.1082108.

[142] Michael Balzer, Andreas Noack, Oliver Deussen, and Claus Lewerentz. Software landscapes: Visualizing the structure of large software systems. In Oliver Deussen, Charles D. Hansen, Daniel A. Keim, and Dietmar Saupe, editors, *VisSym*, pages 261–266. Eurographics Association, 2004. URL http://dblp.uni-trier.de/db/conf/vissym/vissym2004.html#BalzerNDL04.

[143] David N. Card and Robert L. Glass. *Measuring Software Design Quality*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990. ISBN 0-13-568593-1.

[144] Ronald B. Finkbine, Ph.D. Metrics and models in software quality engineering. *SIGSOFT Softw. Eng. Notes*, 21:89–, January 1996. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/381790.565681. URL http://doi.acm.org/10.1145/381790.565681.

[145] I. Brooks. Object-oriented metrics collection and evaluation with a software process. In *Proc. OOPSLA '93 Workshop Processes and Metrics for Object-Oriented Software Development*, Washington, D.C., 1993.

[146] Colin Ware. *Information visualization: perception for design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000. ISBN 1-55860-511-8.

[147] Warwick Irwin and Neville Churcher. Object oriented metrics: Precision tools and configurable visualisations. In *Proceedings of the 9th International Symposium on Software Metrics*, pages 112–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1987-3. URL http://dl.acm.org/citation.cfm?id=942804.943757.

[148] D. Holten, R. Vliegen, and J. J. van Wijk. Visual realism for the visualization of software metrics. In *Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT '05, pages 12–, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9540-9. doi: http://dx.doi.org/10.1109/VISSOF.2005.1684299. URL http://dx.doi.org/10.1109/VISSOF.2005.1684299.

[149] Michele Lanza and Stéphane Ducasse. Polymetric views-a lightweight visual approach to reverse engineering. *IEEE Trans. Softw. Eng.*, 29:782–795, September 2003. ISSN 0098-5589. doi: 10.1109/TSE.2003.1232284. URL http://dl.acm.org/citation.cfm?id=942594.942788.

[150] Richard Wettel and Michele Lanza. Visualizing software systems as cities. In Jonathan I. Maletic, Alexandru Telea, and Andrian Marcus, editors, *VISSOFT*, pages 92–99. IEEE Computer Society, 2007. ISBN 1-4244-0600-5.

[151] M. M. Lehman and L. A. Belady, editors. *Program evolution: processes of software change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985. ISBN 0-12-442440-6.

[152] Alexandru Telea and David Auber. Code flows: Visualizing structural evolution of source code. *Comput. Graph. Forum*, 27(3):831–838, 2008. doi: http://dx.doi.org/10.1111/j.1467-8659.2008.01214.x.

[153] Danny Holten and Jarke J. van Wijk. Visual comparison of hierarchically organized data. *Computer Graphics Forum*, 27(3):759–766, 2008. URL http://dblp.uni-trier.de/db/journals/cgf/cgf27.html#HoltenW08.

[154] Stephen G. Eick, Joseph L. Steffen, and Eric E. Sumner, Jr. Seesoft – a tool for visualizing line oriented software statistics. In Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors, *Readings in information visualization*, pages 419–430. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. ISBN 1-55860-533-9. URL http://dl.acm.org/citation.cfm?id=300679.300788.

[155] Lucian Voinea, Alexandru Telea, and Michel R. V. Chaudron. Version-centric visualization of code evolution. In Ken Brodlie, David J. Duke, and Kenneth I. Joy, editors, *EuroVis*, pages 223–230. Eurographics Association, 2005. ISBN 3-905673-19-3. URL http://dblp.uni-trier.de/db/conf/vissym/eurovis2005.html#VoineaTC05.

[156] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM symposium on Software visualization*, SoftVis '03, pages 77–ff, New York, NY, USA, 2003. ACM. ISBN 1-58113-642-0. doi: http://doi.acm.org/10.1145/774833.774844. URL http://doi.acm.org/10.1145/774833.774844.

[157] Harald Gall, Karin Hajek, and Mehdi Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 190–, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8779-7. URL http://dl.acm.org/citation.cfm?id=850947.853338.

[158] Andreas Kerren and Ilir Jusufi. Novel visual representations for software metrics using 3d and animation. In Jürgen Münch and Peter Liggesmeyer, editors, *Software Engineering (Workshops)*, volume 150 of *LNI*, pages 147–154. GI, 2009. ISBN 978-3-88579-244-4.

[159] Margaret-Anne Storey, Casey Best, and Jeff Michaud. Shrimp views: An interactive environment for exploring java programs. *The 9th International Workshop on Program Comprehension*, 0:111–112, 2001. doi: http://doi.ieeecomputersociety.org/10.1109/WPC.2001.921719.

[160] Marco D'Ambros and Michele Lanza. Bugcrawler: Visualizing evolving software systems. In *11th European Conference on Software Maintenance and Reengineering, 2007. CSMR '07.*, pages 333 –334, march 2007. doi: 10.1109/CSMR.2007.17.

[161] Amit P. Sawant and Naveen Bali. Diffarchviz: A tool to visualize correspondence between multiple representations of a software architecture. *Visualizing Software for Understanding and Analysis, International Workshop on*, 0:121–128, 2007. doi: http://doi.ieeecomputersociety.org/10.1109/VISSOF.2007.4290710.

[162] Andrew McNair, Daniel M. German, and Jens Weber-Jahnke. Visualizing software architecture evolution using change-sets. In *Proceedings of the 14th Working Conference on Reverse Engineering*, pages 130–139, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3034-6. doi: 10.1109/WCRE.2007.52. URL http://dl.acm.org/citation.cfm?id=1339262.1339487.

[163] K. Delhi Babu, P. Govindarajulu, and A.N. Aruna Kumari Ahmed. Development of the conceptual tool for complete software architecture visualization: Darch (da). *International Journal of Computer Science and Network Security (IJCSNS)*, 9(4):277–286, April 2009.

[164] Richard A. Becker and William S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, May 1987. ISSN 0040-1706. doi: 10.2307/1269768. URL http://dx.doi.org/10.2307/1269768.

[165] Matej Novotny and Helwig Hauser. Outlier-preserving focus+context visualization in parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):893–900, September 2006. ISSN 1077-2626. doi: 10.1109/TVCG.2006.170. URL http://dx.doi.org/10.1109/TVCG.2006.170.

[166] Antonio Gonzalez, Roberto Theron, Alexandru Telea, and Francisco J. Garcia. Combined visualization of structural and metric information for software evolution analysis. In *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, IWPSE-Evol '09, pages 25–30, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-678-6. doi: 10.1145/1595808.1595815. URL http://doi.acm.org/10.1145/1595808.1595815.

[167] Helmut Doleisch. SIMVIS: interactive visual analysis of large and time-dependent 3d simulation data. In Shane G. Henderson, Bahar Biller, Ming-Hua Hsieh, John Shortle, Jeffrey D. Tew, and Russell R. Barton, editors, *Proceedings of the Winter Simulation Conference, WSC 2007, Washington, DC, USA, December 9-12, 2007*, pages 712–720. WSC, 2007. ISBN 1-4244-1306-0. doi: 10.1145/1351542.1351674. URL http://doi.acm.org/10.1145/1351542.1351674.

[168] Chris North and Ben Shneiderman. Snap-together visualization: A user interface for coordinating visualizations via relational schemata. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 128–135, New York, NY, USA, 2000. ACM. ISBN 1-58113-252-2. doi: 10.1145/345513.345282. URL http://doi.acm.org/10.1145/345513.345282.

[169] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41(1):2:1–2:31, January 2009. ISSN 0360-0300. doi: 10.1145/1456650.1456652. URL http://doi.acm.org/10.1145/1456650.1456652.

[170] Allen R. Martin and Matthew O. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proceedings of the 6th Conference on Visualization '95*, VIS '95, pages 271–, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7187-4. URL http://dl.acm.org/citation.cfm?id=832271.833844.

[171] Helmut Doleisch, Martin Gasser, and Helwig Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proceedings of the Symposium on Data Visualisation 2003*, VISSYM '03, pages 239–248, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-698-6. URL http://dl.acm.org/citation.cfm?id=769922.769949.

[172] Chris Weaver. Conjunctive visual forms. *IEEE Trans. Vis. Comput. Graph.*, 15(6):929–936, 2009. URL http://dblp.uni-trier.de/db/journals/tvcg/tvcg15.html#Weaver09.

[173] Karim Dhambri, Houari A. Sahraoui, and Pierre Poulin. Visual detection of design anomalies. In *12th European Conference on Software Maintenance and Reengineering, CSMR 2008, April 1-4, 2008, Athens, Greece*, pages 279–283. IEEE Computer Society, 2008. ISBN 978-1-4244-2157-2. URL http://dblp.uni-trier.de/db/conf/csmr/csmr2008.html#DhambriSP08.

[174] Michele Lanza and Stéphane Ducasse. Codecrawler — an extensible and language independent 2d and 3d software visualization tool. In *Tools for Software Maintenance and Reengineering*, pages 74–94. Franco Angeli, 2005. URL http://scg.unibe.ch/archive/papers/Lanz05bCCBookChapter.pdf.

[175] Coen De Roover, Carlos Noguera, Andy Kellens, and Viviane Jonckers. The SOUL tool suite for querying programs in symbiosis with Eclipse. In Christian W. Probst and Christian Wimmer, editors, *PPPJ*, pages 71–80. ACM, 2011. ISBN 978-1-4503-0935-6. URL http://dblp.uni-trier.de/db/conf/pppj/pppj2011.html#RooverNKJ11.

[176] Florian Deissenboeck, Markus Pizka, and Tilman Seifert. Tool support for continuous quality assessment. In Kostas Kontogiannis, Ying Zou, and Massimiliano Di Penta, editors, *13th International Workshop on Software Technology and Engineering Practice (STEP 2005), 24-25 September 2005, Budapest, Hungary*, pages 127–136. IEEE Computer Society, 2005. ISBN 0-7695-2639-X. doi: 10.1109/STEP.2005.31. URL http://doi.ieeecomputersociety.org/10.1109/STEP.2005.31.

[177] Jens Heidrich and JÃijrgen MÃijnch. Goal-oriented customization of software cockpits. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(5):381–405, 2010. ISSN 1532-0618. doi: 10.1002/smr.458. URL http://dx.doi.org/10.1002/smr.458.

[178] T.A. Corbi. Program Understanding: Challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306, 1989.

[179] Florian Holzschuher and René Peinl. Performance of graph query languages: Comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 195–204, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1599-9. doi: 10.1145/2457317.2457351. URL http://doi.acm.org/10.1145/2457317.2457351.

[180] Adrian Silvescu, Doina Caragea, and Anna Altramentov. Graph databases. Technical report, Artificial Intelligence Reseach Laboratory, Iowa State University, 2002.

[181] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0064-3. doi: 10.1145/1900008.1900067. URL http://doi.acm.org/10.1145/1900008.1900067.

[182] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL http://doi.acm.org/10.1145/1656274.1656278.

[183] P Shannon, A Markiel, O Ozier, N S Baliga, J T Wang, D Ramage, N Amin, B Schwikowski, and T Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, November 2003. doi: 10.1101/gr.1239303.

[184] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, June 1994. ISSN 0098-5589. doi: 10.1109/32.295895. URL http://dx.doi.org/10.1109/32.295895.

[185] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen. An empirical study on object-oriented metrics. In *Proceedings of the 6th International Symposium on Software Metrics*, METRICS '99, pages 242–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0403-5. URL http://dl.acm.org/citation.cfm?id=520792.823979.

[186] Robert Martin. OO Design Quality Metrics - An Analysis of Dependencies. In *Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics*. OOPSLA'94, 1994. URL http://www.objectmentor.com/resources/articles/oodmetrc.pdf.

[187] Jagdish Bansiya and Carl G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Trans. Softw. Eng.*, 28(1): 4–17, January 2002. ISSN 0098-5589. doi: 10.1109/32.979986. URL http://dx.doi.org/10.1109/32.979986.

[188] Thomas J. McCabe. A complexity measure. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press. URL http://dl.acm.org/citation.cfm?id=800253.807712.

[189] Viswanath Venkatesh, Michael G. Morris, Gordon B. Davis, and Fred D. Davis. User acceptance of information technology: toward a unified view. *MIS Q.*, 27(3):425–478, September 2003. ISSN 0276-7783. URL http://dl.acm.org/citation.cfm?id=2017197.2017202.

[190] S. Nestler, E. Artinger, T. Coskun, Y. Yildirim, S. Schumann, M. Maehler, F. Wucholt, S. Strohschneider, and G. Klinker. Assessing qualitative usability in life-threatening, time-critical and unstable situations. *GMS Med Inform Biom Epidemiol 2011*, 7(1), 2011.

[191] Taimur Khan, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. Visualization and Evolution of Software Architectures. In Christoph Garth, Ariane Middel, and Hans Hagen, editors, *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011*, volume 27 of *OpenAccess Series in Informatics (OASIcs)*, pages 25–42, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-46-0. doi: http://dx.doi.org/10.4230/OASIcs.VLUDS.2011.25. URL http://drops.dagstuhl.de/opus/volltexte/2012/3739.

[192] Jens Knodel, Dirk Muthig, Matthias Naab, and Mikael Lindvall. Static evaluation of software architectures. In *Proceedings of the Conference on Software Maintenance and Reengineering*, CSMR '06, pages 279–294, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2536-9. URL http://dl.acm.org/citation.cfm?id=1116163.1116412.

[193] Rainer Koschke and Daniel Simon. Hierarchical reflexion models. In *Proceedings of the 10th Working Conference on Reverse Engineering*, WCRE '03, pages 36–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2027-8. URL http://dl.acm.org/citation.cfm?id=950792.951359.

[194] Mathias Naab. Evaluation of graphical elements and their adequacy for the visualization of software architectures. Thesis IESE Report 078.05/E, Fraunhofer IESE, Kaiserslautern, Germany, 2005. URL http://publica.fraunhofer.de/eprints/urn:nbn:de:0011-n-344671.pdf.

[195] Jens Knodel, Dirk Muthig, and Matthias Naab. An experiment on the role of graphical elements in architecture visualization. *Empirical Software Engineering*, 13(6):693–726, 2008.

[196] L.V. Hedges and I. Olkin. *Statistical Method for Meta-Analysis*. Acad. Press, 1985. ISBN 9780123363817. URL http://books.google.de/books?id=brNpAAAAMAAJ.

[197] Jacob Cohen. A power primer. *Psychological Bulletin*, 112:155–159, 1992.

[198] Nelson Wong, Sheelagh Carpendale, and Saul Greenberg. Edgelens: an interactive method for managing edge congestion in graphs. In *Proceedings of the Ninth annual IEEE conference on Information visualization*, INFOVIS'03, pages 51–58, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7803-8154-8.

[199] Nelson Wong and Sheelagh Carpendale. Interactive poster: Using edge plucking for interactive graph exploration. In *Poster in the IEEE Symposium on Information Visualization*, 2005.

[200] Craig Anslow, Stuart Marshall, James Noble, and Robert Biddle. Sourcevis: Collaborative software visualization for co-located environments. In Alexandru Telea, Andreas Kerren, and Andrian Marcus, editors, *VISSOFT*, pages 1–10. IEEE, 2013. ISBN 978-1-4799-1457-9. URL http://dblp.uni-trier.de/db/conf/vissoft/vissoft2013.html#AnslowMNB13.

[201] Matthias Deller, Sebastian Thelen, Daniel Steffen, Peter-Scott Olech, Achim Ebert, Jan Malburg, and Jörg Meyer. A Highly Scalable Rendering Framework for Arbitrary Display and Display-in-Display Configurations. In Hamid R. Arabnia and Leonidas Deligiannidis, editors, *CGVR*, pages 164–170. CSREA Press, 2009. ISBN 1-60132-097-3. URL http://dblp.uni-trier.de/db/conf/cgvr/cgvr2009.html#DellerTSOEMM09.

[202] Tao Ni, Greg S. Schmidt, Oliver G. Staadt, Mark A. Livingston, Robert Ball, and Richard May. A Survey of Large High-Resolution Display Technologies, Techniques, and Applications. In *VR '06: Proceedings of the IEEE conference on Virtual Reality*, pages 223–236, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 1-4244-0224-7. doi: http://dx.doi.org/10.1109/VR.2006.20.

[203] Bruno Raffin and Luciano Soares. PC Clusters for Virtual Reality. In *VR '06: Proceedings of the IEEE conference on Virtual Reality*, pages 215–222, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 1-4244-0224-7. doi: http://dx.doi.org/10.1109/VR.2006.107.

[204] Munjae Song, Seongwon Park, and Yongbin Kang. A Survey on Projector-Based PC Cluster Distributed Large Screen Displays and Shader Technologies. In Hamid R. Arabnia, editor, *CGVR*, pages 153–159. CSREA Press, 2007. ISBN 1-60132-028-0.

[205] J. Whitehead. Collaboration in software engineering: A roadmap. In *Future of Software Engineering, 2007. FOSE '07*, pages 214–225, May 2007. doi: 10.1109/FOSE.2007.4.

[206] Filippo Lanubile, Christof Ebert, Rafael Prikladnicki, and Aurora Vizcaino. Collaboration tools for global software engineering. *IEEE Software*, 27(2):52–55, 2010. ISSN 0740-7459. doi: http://doi.ieeecomputersociety.org/10.1109/MS.2010.39.

[207] Andrew J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 344–353, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2828-7. doi: 10.1109/ICSE.2007.45. URL http://dx.doi.org/10.1109/ICSE.2007.45.

[208] M. Storey, C. Bennett, R.I Bull, and D.M. German. Remixing visualization to support collaboration in software maintenance. In *Frontiers of Software Maintenance, 2008. FoSM 2008.*, pages 139–148, Sept 2008. doi: 10.1109/FOSM.2008.4659257.

[209] Greg Johnson. Collaborative visualization 101. In Richard J. Beach, editor, *SIGGRAPH*, pages 8–11. ACM, 1988. ISBN 0-89791-275-6.

[210] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.*, 21(3):693–702, 2002. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/566654.566639.

[211] Luciano P. Soares and Marcelo K. Zuffo. JINX: an X3D browser for VR immersive simulation based on clusters of commodity computers. In *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*, pages 79–86, New York, NY, USA, 2004. ACM. ISBN 1-58113-845-8. doi: http://doi.acm.org/10.1145/985040.985052.

[212] Dirk Reiners. *OpenSG: A Scene Graph System for Flexible and Efficient Realtime Rendering for Virtual and Augmented Reality Applications*. PhD thesis, Technischen Universität Darmstadt, 2002.

[213] Benjamin Schaeffer and Camille Goudeseune. Syzygy: Native PC Cluster VR. *Virtual Reality Conference, IEEE*, 0:15, 2003. ISSN 1087-8270. doi: http://doi.ieeecomputersociety.org/10.1109/VR.2003.1191116.

[214] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)*, page 89, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-0948-7.

[215] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The CAVE: audio visual experience automatic virtual environment. *Commun. ACM*, 35(6):64–72, 1992. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/129888.129892.

[216] Linus Valtersson. Examination of the possibility to use OpenSceneGraph for real-time graphics using Immersive Projection Technology. In *SIGRAD'03. The Annual SIGRAD Conference. Special Theme: Real Time Simulations*, pages 41–44, Linköping, Sweden, 2003. Linköping University Electronic Press. ISBN 91-7373-797-6. doi: http://www.ep.liu.se/ecp/010/.

[217] Allen Bierbaum and Carolina Cruz-Neira. ClusterJuggler: A modular architecture for immersive clustering. In *Workshop on Commodity Clusters for Virtual Reality, IEEE Virtual Reality Conference*, 2003.

[218] Han Chen, Douglas W. Clark, Zhiyan Liu, Grant Wallace, Kai Li, and Yuqun Chen. Software Environments For Cluster-Based Display Systems. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 202, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1010-8.

[219] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A Sorting Classification of Parallel Rendering. *IEEE Comput. Graph. Appl.*, 14(4):23–32, 1994. ISSN 0272-1716. doi: http://dx.doi.org/10.1109/38.291528.

[220] Otmar Hilliges, Lucia Terrenghi, Sebastian Boring, David Kim, Hendrik Richter, and Andreas Butz. Designing for collaborative creative problem solving. In *C&C '07: Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, pages 137–146, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-712-4. doi: http://doi.acm.org/10.1145/1254960.1254980.

[221] S. Vinoski. CORBA: integrating diverse applications within distributed heterogeneous environments. *Communications Magazine, IEEE*, 35(2):46–55, Feb 1997. ISSN 0163-6804. doi: 10.1109/35.565655.

[222] Judith S. Dahmann, Frederick S. Kuhl, and Richard M. Weatherly. Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation. *Simulation*, 71(6):378–387, 1998.

[223] Yasmin I. Al-Zokari, Taimur Khan, Daniel Schneider, Dirk Zeckzer, and Hans Hagen. CakES: Cake Metaphor for Analyzing Safety Issues of Embedded Systems. In Hans Hagen, editor, *Scientific Visualization: Interactions, Features, Metaphors*, volume 2 of *Dagstuhl Follow-Ups*, pages 1–16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2011. ISBN 978-3-939897-26-2. doi: http://dx.doi.org/10.4230/DFU.Vol2.SciViz.2011.1. URL http://drops.dagstuhl.de/opus/volltexte/2011/3284.

[224] N. Limnios. *Fault Trees (Control Systems, Robotics & Manufacturing Series)*. Wiley, John & Sons, 2007.

[225] B. Kaiser, C. Gramlich, and M. Foerster. State/event fault trees - safety analysis model for software-controlled systems. *Reliability engineering & systems safety*, 92:1521–1537, 2007. doi: DOI:10.1016/j.ress.2006.10.010.

[226] P. Gelder T. Bedford. *Safety and Reliability : Proceedings of the ESREL 2003 Conference, Maastricht the Netherlands, 15-18 June 2003*. Taylor & Francis, 2003.

[227] Yasmin I. Al-Zokari, Daniel Schneider, Dirk Zeckzer, Liliana Guzman, Yarden Livnat, and Hans Hagen. Enhanced cakes representing safety analysis results of embedded systems. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *Federated Conference on Computer Science and Information Systems - FedCSIS 2011, Szczecin, Poland, 18-21 September 2011, Proceedings*, pages 783–790, 2011. ISBN 978-83-60810-22-4. URL http://fedcsis.eucip.pl/proceedings/pliks/50.pdf.

[228] Ragaad AlTarawneh, Jens Bauer, and Achim Ebert. A visual interactive environment for enhancing collaboration between engineers for the safety analysis mechanisms in embedded systems. In Susan R. Fussell, Wayne G. Lutters, Meredith Ringel Morris, and Madhu Reddy, editors, *Computer Supported Cooperative Work, CSCW '14, Baltimore, MD, USA, February 15-19, 2014, Companion Volume*, pages 125–128. ACM, 2014. ISBN 978-1-4503-2541-7. doi: 10.1145/2556420.2556480. URL http://doi.acm.org/10.1145/2556420.2556480.

[229] David Lorge Parnas. Software aging. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. ISBN 0-8186-5855-X. URL http://dl.acm.org/citation.cfm?id=257734.257788.

[230] Andy Zaidman. Scalability solutions for program comprehension through dynamic analysis. In *CSMR*, pages 327–330. IEEE Computer Society, 2006. ISBN 0-7695-2536-9. URL http://dblp.uni-trier.de/db/conf/csmr/csmr2006.html#Zaidman06.

[231] Helwig Hauser. Interactive visual analysis - an opportunity for industrial simulation. In Thomas Schulze, Graham Horton, Bernhard Preim, and Stefan Schlechtweg, editors, *SimVis*, pages 1–6. SCS Publishing House e.V., 2006. ISBN 3-936150-46-X. URL http://dblp.uni-trier.de/db/conf/simvis/simvis2006.html#Hauser06.

[232] Marco D'Ambros. *On the Evolution of Source Code and Software Defects*. CreateSpace, Paramount, CA, 2012. ISBN 1460953568, 9781460953563.

[233] John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, Cambridge, MA, 1998. ISBN 978-0-262-19395-5.

[234] Giuseppe A. Di Lucca and Massimiliano Di Penta. Experimental settings in program comprehension: Challenges and open issues. In *ICPC*, pages 229–234. IEEE Computer Society, 2006. ISBN 0-7695-2601-2. URL http://dblp.uni-trier.de/db/conf/iwpc/icpc2006.html#LuccaP06.

[235] Geoffrey Ellis and Alan Dix. An explorative analysis of user evaluation studies in information visualisation. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, pages 1–7, New York, NY, USA, 2006. ACM. ISBN 1-59593-562-2. doi: 10.1145/1168149.1168152. URL http://doi.acm.org/10.1145/1168149.1168152.

[236] Mariam Sensalire, Patrick Ogao, and Alexandru Telea. Evaluation of software visualization tools: Lessons learned. In *VISSOFT*, pages 19–26. IEEE Computer Society, 2009. URL http://dblp.uni-trier.de/db/conf/vissoft/vissoft2009.html#SensalireOT09.

[237] Steven P. Reiss. The paradox of software visualization. In Stéphane Ducasse, Michele Lanza, Andrian Marcus, Jonathan I. Maletic, and Margaret-Anne D. Storey, editors, *VISSOFT*, pages 59–63. IEEE Computer Society, 2005. ISBN 0-7803-9540-9. URL http://dblp.uni-trier.de/db/conf/vissoft/vissoft2005.html#Reiss05.

[238] Alex Endert, M. Shahriar Hossain, Naren Ramakrishnan, Chris North, Patrick Fiaux, and Christopher Andrews. The human is the loop: new directions for visual analytics. *Journal of Intelligent Information Systems*, pages 1–25, 01/2014 2014. ISSN 0925-9902. doi: 10.1007/s10844-014-0304-9. URL http://dx.doi.org/10.1007/s10844-014-0304-9.

[239] Y. Hassan-Montero and V. Herrero-Solana. Improving Tag-Clouds as Visual Information Retrieval Interfaces. In *Proc. InSciT 2006*, Merida, Spain, October 2006.

[240] Yarden Livnat, James Agutter, Shaun Moon, and Stefano Foresti. Visual correlation for situational awareness. In John T. Stasko and Matthew O. Ward, editors, *INFOVIS*, page 13. IEEE Computer Society, 2005. ISBN 0-7803-9464-X. URL http://dblp.uni-trier.de/db/conf/infovis/infovis2005.html#LivnatAMF05.

Appendix

A

List of Acronyms

AST	Abstract Syntax Tree
API	Application Program Interface
AVT	Architecture Visualization Tool
BE	Basic Event
CAG	Compound Attributed Graph
CakES	Cake metaphor for safety analysis of Embedded Systems
CDT	C/C++ Development Toolkit
CG	Computer Graphics
CMV	Coordinated Multiple Views
CORBA	Common Object Request Broker Architecture
CU	Compilation Unit
DAG	Directed Acyclic Graph
DBIS	Databases and Information Systems
FT	Fault Tree
DIT	Depth of Inheritance Tree
eCITY	Evolving CITY
eCITY+	Evolving CITY-plus
FOAF	Friend Of A Friend
HCI	Human Computer Interaction
HEB	Hierarchical Edge Bundle
Hi-Res	High-Resolution
HLA	High Level Architecture
IDE	Integrated Development Environment
IESE	Institute for Experimental Software Engineering
IRTG	International Research Training Group 1131
IVA	Interactive Visual Analysis

JDT	Java Development Tooling
KNIME	Konstanz Information Miner
LOC	Lines of Code
LC	Lines of Comment
MCS	Minimal Cut Set
MPI	Message Passing Interface
NoSQL	Not Only SQL
NumOfLMs	Number of Local Methods
NumOfMAs	Number of Method Accesses
RAVON	Robust Autonomous Vehicle for Off-road Navigation
RFC	Response For a Class
SAVE	Software Architecture Visualization and Evaluation
SC	Statement Count
SCM	Source Control Management
SDK	Software Development Kit
SDL	Simple DirectMedia Layer
SE	Software Engineering
SPCC	Software Project Control Center
SQL	Structured Query Language
SML	Service Modeling Language
TAM	Technology Acceptance Model
UML	Unified Modeling Language
VA	Visual Analytics
VIMETRIK	Visual Specification of Metrics
VPL	Visual Programming Language
VR	Virtual Reality
VRML	Virtual Reality Modeling Language

B

Declaration of Authorship

I, **Taimur Kausar Khan**, declare that this dissertation titled, 'Interactive Visual Analysis of Software Structures' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a doctoral degree at the Technische Universität Kaiserslautern
- No part of this dissertation has been submitted previously, in whole or in part, to qualify for any other academic award
- No other person's work has been used without due acknowledgement in this dissertation
- All references and verbatim extracts have been quoted, and all sources of information, including graphs and data sets, have been specifically acknowledged

Kaiserslautern, 20-Jul-2015 *Taimur Khan*

C

Curriculum Vitae

Taimur Khan

Curriculum Vitae

Ziegelstrasse 47
Kaiserslautern, Germany 67655
☎ (0631) 205 4562
✉ tkhan@informatik.uni-kl.de

"You can have data without information, but you cannot have information without data" - Daniel Keys Morgan

Professional Summary

A Ph.D. researcher in the area of Visualization and Computer Graphics at the Technische Universität Kaiserslautern. Experience includes six years of research in the area of Software Visualization and Information Visualization with a focus on Visual Analytics in Software Engineering – bridging the gap between theory and practice. Experience also includes technical writing, technical presentation at conferences and exhibitions, and three years of university teaching and administration. Expected thesis defense date: 17.07.2015.

Education

2009–Present **Doctor of Engineering**, *Technische Universität Kaiserslautern*, Germany.
Specialization: Visual Analytics in Software Engineering (Expected Graduation: July 2015)

Dissertation: *Interactive Visual Analysis of Software Structures*
In collaboration with the Fraunhofer IESE, my dissertation investigates novel visual analytics strategies and methodologies for software quality and maintenance. The focus of this work is to provide software quality experts with an easier means to access software data, a more intuitive means to generate software measurements, and importantly, an integrated means to visualize these measurement results.

2000–2002 **Masters of Science**, *Old Dominion University*, Virginia, USA, *GPA – 3.5.*
Specialized in Computer Engineering

Master's Thesis: *Approaching High Fidelity Human Tracking and Representation in Interactive Virtual Environments*
This thesis investigates the role of immersive (CAVE) and non-immersive (3D Desktop) environments for the training of personnel manning a security checkpoint. In order to conduct a comparative study, the issues of effectively tracking, recording, and replaying movements, events, and actions are addressed.

1996–2000 **Bachelor of Science**, *Old Dominion University*, Virginia, USA, *GPA – 3.7.*
Specialized in Computer Engineering

Research Interests

My principal research interests lie in the fields of human-computer interaction and visual analytics, specifically towards solutions that assist in the sense-making of real-world data. I am currently investigating the effects of a tighter coupling between software data mining and software visualization on software understanding and maintenance.

My future research plans are to build on the foundations of my Ph.D. to further investigate the application of human-computer interaction and visual analytics on different fields and domains, with a view to bridge the gap between academia and industry. In addition, I am quite excited at the prospect of new and emerging fields such as *Big Data* and *Data Science*. These fields present enormous challenges, thanks to the relentless increase in the volume, velocity, and variety of information ripe for mining and analysis.

Research Experience

2009–Present **Research Assistant**, Technische Universität Kaiserslautern, Germany.
Research focus has been on bridging the gap between the theory and research of Software Visualization. Ultimately, the goal of this work has been to reduce software maintenance costs through novel interactive exploratory visualization concepts and to apply these modern techniques in the context of services offered by software quality analysis.

Detailed research achievements:

- Develop methodologies that combine computational analysis methods together with sophisticated visualization methods and metaphors through an interactive visual analysis approach
- Introduce a novel approach for the visual analysis of software measurement data that captures complete facts of the system, employs a flow-based visual paradigm for the specification of software measurement queries, and presents measurement results through integrated software visualizations
- Extend existing tools with additional views of the data for the presentation and interactive exploration of system artifacts and their inter-relations
- Work in coordination with the Fraunhofer IESE to develop prototype tools to evaluate if these tools indeed improve the understanding of large and complex software systems. Responsibilities include: initial concept and design, programming and technical implementation, and quality assurance and comprehensive testing

2000–2004 **Graduate Research Assistant**, VMASC, Virginia, USA.
Design, development, and implementation of human computer interface and simulation components in both immersive (CAVE) and non-immersive (3D Desktop) environments.

Teaching and Administrative Experience

2005–2008 **CIT Faculty and coordinator**, American College of Dubai, Dubai, UAE.
Responsible for various courses, student counseling, and placing students as interns. Additional responsibilities include proposal and accreditation for an undergraduate degree in the field of Computer Information Technology (CIT).

Courses Taught: Introduction to IT & Applications, General Mathematics, Introduction to Programming, and Introduction to Communication Technology

2005–2008 **Adjunct Lecturer**, American University in Dubai, Dubai, UAE.
Responsible for introductory computer courses as part of the Liberal Arts curriculum.

Courses Taught: Introduction to Computers, The Internet, and Development Mathematics

D

List of Own Publications

Forthcoming

[1] **Taimur Khan**, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. Visual Analytics of Software Structure and Metrics. To appear in the *3rd IEEE Working Conference on Software Visualization (VISSOFT' 2015)*. Bremen, Germany, September 2015.

[2] **Taimur Khan**, Henning Barthel, Karsten Amrhein, Achim Ebert, and Peter Liggesmeyer. An Interactive Approach for Inspecting Software System Measurements. To appear in the *15th IFIP TC.13 International Conference on Human-Computer Interaction (INTERACT' 2015)*. Bamberg, Germany, September 2015.

Book Chapter

[1] **Taimur Khan**, Henning Barthel, Liliana Guzman, Achim Ebert, and Peter Liggesmeyer. eCITY: Evolutionary Software Architecture Visualization – An Evaluation. In Achim Ebert, Gerrit C. van der Veer, Gitta Domik, Nahum D. Gershon, and Inga Scheler, editors, *Building Bridges: HCI, Visualization, and Non-formal Modeling*, Lecture Notes in Computer Science, pages 201–224. Springer Berlin Heidelberg, 2014. ISBN 978-3-642-54893-2. doi: 10.1007/978-3-642-54894-9_15. URL http://dx.doi.org/10.1007/978-3-642-54894-9_15.

Conference Proceedings

[1] **Taimur Khan**, Shah Rukh Humayoun, Karsten Amrhein, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. eCITY+: a Tool to Analyze Software Architectural Relations Through Interactive Visual Support. In *Proceedings of the 2014 European Conference on Software Architecture Workshops (ECSAW' 2014)*, pages 361–364, Vienna, Austria, August 2014. ISBN 978-1-4503-2778-7. doi: 10.1145/2642803.2642839. URL http://doi.acm.org/10.1145/2642803.2642839.

[2] **Taimur Khan**, Shah Rukh Humayoun, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. eCITY+: a Visual Environment for Analysing Software Structure and Evolution. In *Demo Proceedings of the 12th International Conference on Advanced Visual Interfaces (AVI' 2014)*, Como, Italy, May 2014.

[3] **Taimur Khan**, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. Visual Exploration of Architectural Metric Data Evolution. In *MetriKon 2013 - Praxis der Software-Messung : Tagungsband des DASMA Software Metrik Kongresses*. Magdeburger Schriften zum Empirischen Software Engineering, Kaiserslautern, Germany, November 2013.

[4] **Taimur Khan**, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. eCITY: a Tool to Track Software Structural Changes Using an Evolving City. In Proceedings of the *29th IEEE International Conference on Software Maintenance (ICSM' 2013)*, pages 492–495. Eindhoven, The Netherlands, September 2013.

[5] **Taimur Khan**, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. Software Architecture Visualization in Theory and Practice. In *Workshop Proceedings of the 30th ACM European Conference on Cognitive Ergonomics (ECCE' 2012)*, Edinburgh, United Kingdom, August 2012.

Peer-reviewed Publications

[1] **Taimur Khan**, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. Visualization and Evolution of Software Architectures. In Christoph Garth, Ariane Middel, and Hans Hagen, editors, *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011*, volume 27 of *OpenAccess Series in Informatics (OASIcs)*, pages 25–42. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2012. ISBN 978-3-939897-46-0. doi: http://dx.doi.org/10.4230/OASIcs.VLUDS.2011.25. URL http://drops.dagstuhl.de/opus/volltexte/2012/3739.

[2] **Taimur Khan**, Daniel Schneider, Yasmin Al-Zokari, Dirk Zeckzer, and Hans Hagen. Framework for Comprehensive Size and Resolution Utilization of Arbitrary Displays. In Hans Hagen, editor, *Scientific Visualization: Interactions, Features, Metaphors*, volume 2 of *Dagstuhl Follow-Ups*, pages 144–159. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2011. ISBN 978-3-939897-26-2. doi: http://dx.doi.org/10.4230/DFU.Vol2.SciViz.2011.144. URL http://drops.dagstuhl.de/opus/volltexte/2011/3291.

[3] **Taimur Khan**. A Survey of Interaction Techniques and Devices for Large High Resolution Displays. In Ariane Middel, Inga Scheler, and Hans Hagen, editors, *Visualization of Large and Unstructured Data Sets - Applications in Geospatial Planning, Modeling and Engineering (IRTG 1131 Workshop)*, volume 19 of *OpenAccess Series in Informatics (OASIcs)*, pages 27–35. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2011. ISBN 978-3-939897-29-3. doi: http://dx.doi.org/10.4230/OASIcs.VLUDS.2010.27. URL http://drops.dagstuhl.de/opus/volltexte/2011/3094.

[4] Yasmin I. Al-Zokari, **Taimur Khan**, Daniel Schneider, Dirk Zeckzer, and Hans Hagen. CakES: Cake metaphor for analyzing safety issues of Embedded Systems. In Hans Hagen, editor, *Scientific Visualization: Interactions, Features, Metaphors*, volume 2 of *Dagstuhl Follow-Ups*, pages 1–16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2011. ISBN 978-3-939897-26-2. doi: http://dx.doi.org/10.4230/DFU.Vol2.SciViz.2011.1. URL http://drops.dagstuhl.de/opus/volltexte/2011/3284.